A NON-INTRUSIVE VIDEO TRACKING METHOD TO MEASURE MOVEMENT OF A MOORED VESSEL

By

Johan Kieviet

Thesis presented in partial fulfilment of the requirements for the degree Master of Science in the Faculty of Civil Engineering at the University of Stellenbosch

Supervisor: Mr Geoff Toms

March 2015

DECLARATION

By submitting this dissertation electronically, I declare that the entirety of the work contained there-in is my own, original work, that I am the sole author thereof (save to the extent explicitly stated otherwise), that reproduction and publication thereof by Stellenbosch University will not infringe any third party rights and that I have not previously in its entirety or in part submitted it for obtaining any qualification.

…….. …….

Johan Kieviet Date

ABSTRACT

There are several ports around the world currently experiencing problems with moored vessel motions. Extreme vessel motions are mainly caused by long waves, which can become trapped inside a harbour basin. The extreme motions can cause downtime in port operations and in some instances cause mooring lines to break.

Methods and procedures currently available to measure motions of moored vessels require vessel specific information as input. The implementation of these methods is seen as impractical to implement on every vessel visiting the port and require the physical measurement of some points on the vessel and/or the placement of some kind of measurement device on the vessel.

A new Six Degree of Freedom (6DOF) motion measurement system for a moored vessel is presented in this document. The system analyses a video image sequence from one camera. The method estimates the 3D rigid motion for an object of known size by using a Pose from Orthography and Scaling with ITerations (POSIT) algorithm. The object for which the motion is estimated is located on the deck of the vessel and within the camera field of view. Geometric rigid body calculations allow for the calculation of camera perspective rotations and translation of an object on the vessel. Further geometric calculations allow for converting camera perspective motions to the 6DOF object motions.

The primary objective of this study was to validate and verify the motions obtained from two proof-of-concept tracking systems. For evaluation purposes, the validation was done by using a small scale physical model set-up in a hydraulics laboratory and using a known method as reference. The Keoship system from the Council for Scientific and Industrial Research (CSIR) is currently one of the most accurate small scale vessel motion measurement systems and was used as reference.

The first method tested was the tracking of a 2D LED rectangle mounted on the vessel. This method tracked a 2D object and was primarily used as a stepping stone to measure movement of a 3D object. The second method tracked a 3D object on the vessel. Each tracking method was tested for four different wave conditions with each condition additionally repeated twice as repeatability tests, resulting in a total of 12 tests for each tracking method.

When comparing the 2D LED tracking and 3D Object tracking data to data measured with the Keoship system, results show that in general, the 3D Object tracking data compared better to the Keoship data. Tests under controlled conditions enabled a direct estimation of the absolute accuracy of the two developed methods.

The verification and accuracy test results, indicated that the 2D LED tracking system should not be pursued further. The results also indicated that for prototype motions exceeding 0.6 m (i.e. storm events) the 3D Object tracking system would have an accuracy close to the maximum allowable accuracy criterion of 0.1 m. This makes the system viable at its current proof-of-concept stage for further development which would enable rapid deployment during a storm event in a prototype situation.

OPSOMMING

Daar is verskeie hawens regoor die wêreld wat tans bewegings probleme op gemeerde skepe ervaar. Hierdie buitensporige bewegings word veroorsaak deur lang periode golwe wat binne die hawe bekkens vasgekeer word. Dit kan daartoe lei dat hawe bedrywighede tot stilstand kom en in ernstige gevalle ook veroorsaak dat meringslyne breek.

Huidige metodes vir die meet van skeepsbewegings op vasgemeerde skepe, vereis skeep spesifieke inligting as inset. Die toepassing van hierdie metodes op elke skip wat die hawe besoek, word as onprakties beskou, aangesien dit die fisiese meting van sekere punte op die skip behels. In sekere gevalle is dit selfs nodig om meet toestelle op die skip te plaas.

In hierdie dokument word ‘n nuwe metode aangebied om die ses grade van vryheid bewegings vir ‘n vasgemeerde skip te meet. Hierdie stelsel analiseer ‘n video beeld reeks van een kamera. Die metode bereken die 3D rigiede beweging van ‘n voorwerp, waarvan die grootte bekend is. ’n ‘Pose from Orthography and scaling with Iterations’ (POSIT) algoritme word hiervoor gebruik. Die voorwerp waarvoor beweging gemeet word is op die dek van die skip en in kamera sig. Rigiede geometriese voorwerp berekeninge word gebruik om die rotasie en translasie vanuit ‘n kamera perspektief te bereken. Verdere geometriese berekeninge maak dit moontlik om die bewegings vanuit die kamera perspektief te omskep in die ses grade van vryheid bewegings van die voorwerp.

Die hoof doelwit van hierdie ondersoek was om die gemete bewegings van twee beweging stelsels te valideer en te verifieer. Die validasie en verifiëring was in ‘n hidrolise laboratorium met ‘n klein skaal model opstelling getoets. ‘n Meet metode van skeepsbeweging op klein skaal wat reeds bekend is, is gebruik as ‘n verwysingsraamwerk waarteen die metings vergelyk kan word. Die Keoship stelsel van die Wetenskaplike Nywerheids Navorsings Raad (WNNR) is tans een van die mees akkurate klein skaal skeepsbeweging meet stelsels, en was as verwysing gebruik.

Die eerste bewegings metode is getoets op ‘n 2D reghoek vervaaridig uit ligstralede diodes. Hierdie metode het die 2D voorwerp gevolg en is hoofsaaklik gebruik as ‘n boublok om die beweging van ‘n 3D voorwerp te volg. Die tweede metode het die beweging van ‘n 3D voorwerp op ‘n skip gevolg. Vir elke meet metode was daar vier verskillende golf toestande. Elke golf toestand was ook ‘n verdere twee keer herhaal vir herhaalbaarheids doeleindes. Saam met die herhaalbaarheids toetse was daar in totaal, 12 toetse vir elkeen van die twee metodes gedoen.

Met die Keoship metode as verwysing, bewys hierdie toetse dat die 3D metode beter resultate lewer as die 2D metode. Toetse onder beheerde toestande, het dit moontlik gemaak om die absolute akkuraatheid van albei sisteme wat ontwikkel was, te evalueer.

Verifikasie en akuraatheids toetse het aangedui dat verdere ontwikkeling van die 2D metode gestuit moet word. Die resultate het ook aangedui dat die 3D metode ‘n akuraatheid baie na aan die maatstaf van 0.1 m sal hê wanneer prototipe bewegings 0.6 m oorskrei (b.v. gedurende ‘n storm). Dit sal die oplossing lewensvatbaar maak by die huidige bewys van konsep fase vir die verdere ontwikkeling wat vinnige ontplooiing gedurende ‘n storm sal moontlik maak.

ACKNOWLEDGEMENTS

I would like to express my gratitude to a number of people who contributed to the successful completion of this thesis and thank the following people:

My employer, the CSIR for financing my studies and allowing me time off from work to attend lectures and finish off my thesis report. Particularly I would like to thank Jatin Harribhai for his efforts and assistance with image processing and training me in the basics of image processing. The hydraulics laboratory staff, Reagan Solomons and Rafick Jappie who assisted with the physical modelling tests, and Gregory Davids who constructed the three dimensional blocks used. Marius Rossouw for his insight in small scale physical modelling.

Geoff Toms my supervisor for his guidance and comments during the entire thesis time.

My parents for the support and encouragement throughout this post graduate course.

I would especially like to thank my wife Lindy for all the support and love.

Johan Kieviet

A NON-INTRUSIVE VIDEO TRACKING METHOD TO MEASURE MOVEMENT OF A MOORED VESSEL

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

1 INTRODUCTION

Several ports around the world are experiencing problems with long wave resonance inside their harbour basins. These long waves are present in all coastal regions experiencing swell and present a challenge to port designers in terms of mitigating port downtime (Stuart 2013). These waves, also known as infragravity waves, are characterized as waves with periods ranging between 30 and 300 seconds with amplitudes generally less than 0.5 m.

Due to the physical properties of long waves, these waves easily penetrate harbour basins that are primarily designed to reduce short wave energy. Furthermore they are typically ignored in generic coastal port design and wave measurement campaigns, leading to a lack of information on long waves at most sites (Stuart 2013). The penetrating long waves can result in vessel motions exceeding the operating limit or design limit of the port, resulting in downtime or even dangerous accidents when mooring lines are forced to snapping point during severe motions.

Current methods and procedures which are available to measure and monitor ship motions for moored vessels require a lot of vessel specific information as input. These methods are seen as intrusive and impractical to implement on every vessel visiting the port.

The topic for this dissertation took root from recently experienced problems with vessels when moored at the Port of Ngqura. The Council for Scientific and Industrial Research (CSIR) went through an elaborate exercise to measure vessel motions during storm events. The exercise required the placement of Differential Global Positioning Systems (DGPS's) on certain vessels during storm events. The resources required in deploying such instruments on vessels as well as the procedure of installation are excessive and require a lot of planning for each individual vessel. The installation location of each Global Positioning System (GPS), and the vessel dimensions as input parameters during post processing, makes this method of measuring vessel motions impractical in the long term.

There is therefore a need for a non-intrusive method with minimum user input requirements that can be used to measure the movements of vessels when moored.

The Hydraulics Laboratory at the CSIR frequently conducts small scale vessel motion studies for various port layout designs and wave conditions. Measurement techniques in the laboratory however, also require the placement of equipment on model vessels representing prototype (full scale) situations.

Recently, optical motion measurements have become the standard for measuring vessel motions in most Hydraulics Laboratories. These optical motion measurement techniques do not require the placement of motion sensors on a vessel. The markers or targets being tracked by the optical camera equipment, do however, still need to be mounted on the vessel. The installation location of each marker, or target, being tracked in vessel coordinates, is then also needed and used to convert tracked movements to vessel motions.

Following these optical approaches, the work presented here-in, aims to show that an image-based method, which does not require the placement of markers or targets on a vessel, can be used to measure vessel motions. This will reduce installation time and costs when measuring moored vessel motions.

This dissertation will focus on verification testing in a controlled environment as well as physical model testing in a hydraulics laboratory. Verification testing will determine the accuracy of the system. Physical model testing will be done to validate such a non-intrusive image-based vision system for measuring the movements of a vessel when berthed. Physical model work will include a comparison with current vessel motion measurement techniques used in the laboratory as well as investigation of the use thereof or a prototype situation.

1.1 Objective and Approach

The objective is to verify the accuracy of a non-intrusive system and to validate the use of a non-intrusive, image-based method for measuring the movement of a moored vessel when berthed.

The approach involves a set-up to verify the accuracy of the system. The verification set-up will be erected in the dry and allow for control measurements for accuracy verification. It also involves the set up and testing of a small scale physical model to validate and compare the new method to those being currently used by the CSIR. All testing will be done at the hydraulics laboratory of the CSIR in Stellenbosch.

1.2 Outline

For the remaining Chapters, a detailed literature review, which includes basic definitions and concepts are provided in Chapter 2. Measurement methods, in prototype, physical modelling domain, and current video tracking methods are also discussed in Chapter 2.

The experiment plan is explained in Chapter 3. The static verification testing done in a workshop, along with the possible measurement errors which may occur during testing, is presented in Chapter 4.

Chapter 5 gives information on the model constructed and the test vessel used while a description on the different measurement devices used in the physical model as well as the set up and calibration information thereof follow in Chapter 6.

Test and data analysis procedures for the physical model testing are explained in Chapter 7.

Chapter 8 presents and also compares the physical model test results from the prototype tracking methods to the Keoship system. The root-mean-square error, R_{rms} is also used to quantify the difference between the data sets

Results for the verification accuracy and physical model tests are discussed in Chapter 9 while a conclusion is provided in Chapter 10.

A possible starting point for future work and a vision from the author on how further research and development on the system would be beneficial to port operations is presented in Chapter 11.

2 LITERATURE REVIEW

2.1 Ocean Waves

It is necessary to understand the classification of wave types in order to identify the various wave types. Ocean surface waves can be classified by a number of properties including wave period/frequency, wave shape, motion, the disturbing force and/or restoring force (Stuart 2013).

Kinsman (1965), gives a possible arrangement which is similar to the bands of the electromagnetic spectrum. His representation is presented in Figure 2.1 and can be used as a visual aid in understanding the classification of waves. Kinsman (1965), also mentions that the curve representing the relative amounts of energy contained in ocean surface waves, is only approximate. However, from this approximate representation we can gather that the band of wind-driven gravity waves possibly contain more wave energy per cycle than any other band.

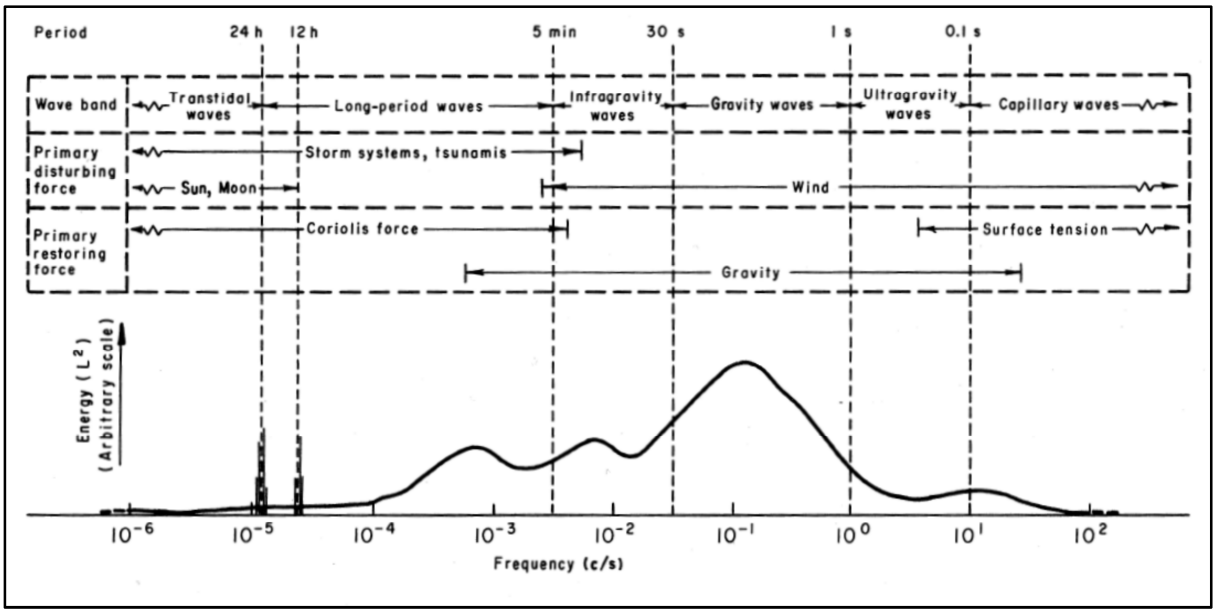

Figure 2.1: Approximate representation of the energy contained in the surface waves of the ocean (Kinsman 1965)

As highlighted earlier, long waves penetrating harbour basins can result in severe vessel motions, causing port downtime and dangerous conditions in the harbour. Figure 2.1 above indicates that infragravity or long waves are between 30s and 300s. For the Port of Ngqura, it is known that infragravity waves occur particularly when storm events approach the South African coast from a south or south westerly direction (Stuart 2013).

2.2 Vessel Motions

The dynamics of a floating rigid body and the surrounding fluid motions are governed by the combined actions of different external forces and moments as well as by the inertia of the bodies themselves. The forces and moments acting on the floating body cannot be considered to be acting at a single point or at discrete points of the system. Instead, they must be regarded as distributed evenly throughout the mass of the fluid particles (Journée & Pinkster 2001). The subsections which follow will describe vessel behaviour, axis conventions for calculating vessel motions, ship motion formulae and the effect of long waves on moored vessels.

2.2.1 Vessel Behaviour

The motions of a vessel, just as for any other three dimensional (3D) rigid body, can be split into three mutually perpendicular translations of the centre of gravity (CoG) and three rotations around the CoG (Journée & Pinkster 2001). These translations and rotations are defined as the six degrees of freedom (6DOF) for a vessel. A definition for these translations and rotations is given below (Journée & Pinkster 2001) along with Figure 2.2 for visual reference:

- The three translations for the CoG in the direction of the x-, y- and z-axes are:
  - Surge in the longitudinal x -direction, positive forwards
  - Sway in the lateral y -direction, positive to port side
  - Heave in the vertical z -direction, positive upwards
- The three rotations about these axes are:
  - Roll about the x -axis, positive right turning
  - Pitch about the y -axis, positive right turning
  - Yaw about the z -axis, positive right turning.

The translations and rotations defined are completely independent and can be used to specify the displaced position and orientation of the vessel in a reference system constrained to the 3D body in space (Benetazzo 2011).

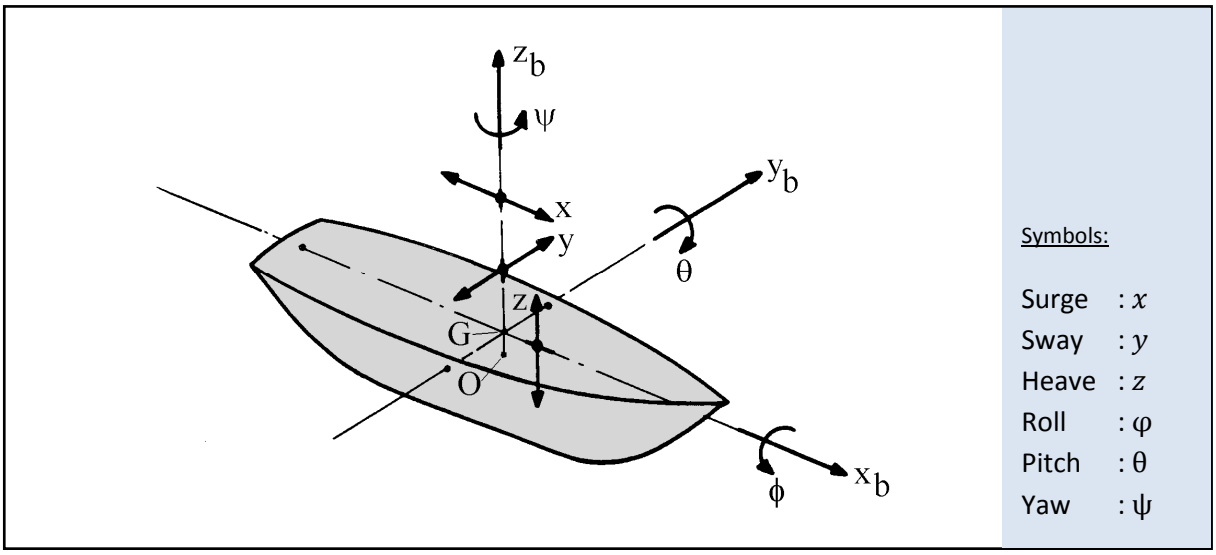

Figure 2.2: Definition of Vessel Motions in Six Degrees of Freedom, where subscript 'b' denotes body-bound coordinate system (x_b, y_b, z_b) (Journée & Pinkster 2001)

2.2.2 Axis conventions

Ship motions are defined by three right-handed orthogonal coordinate systems:

- Earth-bound coordinate system, $S(x_0,\ y_0, z_0)$.
  The $(x_0,\ y_0)$-plane, lies on the still water surface, with the positive x_0 -axis in the direction of the wave propagation. The plane can be rotated at a horizontal angle μ, relative to the translating axis system $O(x,\ y,\ z)$ as shown in Figure 2.3. The positive z_0-axis is directed upwards (Journée & Pinkster 2001).

- Body-bound coordinate system, $G(x_b,\ y_b, z_b)$.
  The origin of the coordinate system is at the ships CoG. The positive axes directions are; x_b in the longitudinal forward direction, y_b in the lateral port side direction and z_b upwards. The $(x_b,\ y_b)$-plane lies parallel to the still water surface (Journée & Pinkster 2001).

- Steadily translating coordinate system, $O(x,\ y,\ z)$.
  This coordinate system is moving with a constant ship speed V. For a stationary vessel, the $O(x,\ y,\ z)$ axes are in the same direction as those of the $G(x_b,\ y_b, z_b)$ axes. The $(x,\ y)$ -plane lies on the still water surface, with its origin at above or under the time-averaged position of the CoG. Ship oscillations are around this steadily translating $O(x,\ y,\ z)$ coordinate system (Journée & Pinkster 2001).

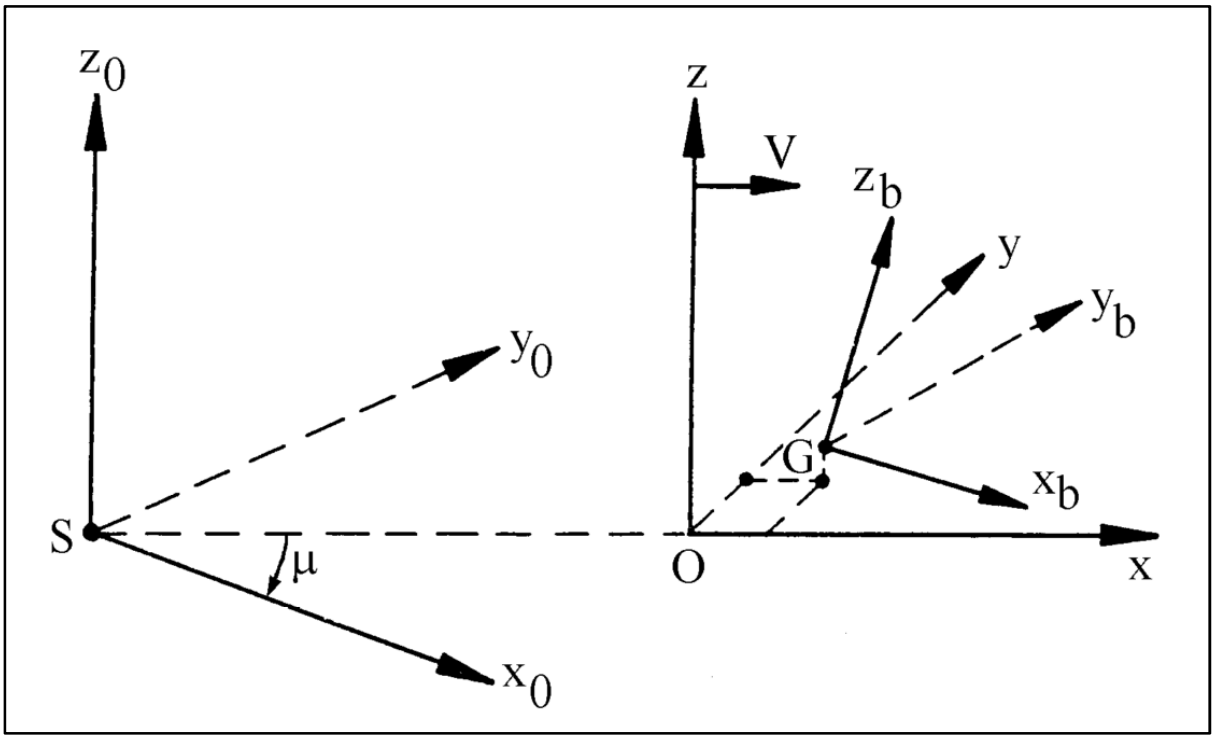

Figure 2.3: Coordinate systems (Journée & Pinkster 2001)

2.2.3 Motions superposition

For the steadily translating $O(x,\ y,\ z)$ system, ship motions are defined by three translations of its CoG and the three rotations given in Figure 2.2. Knowing the motions of and around the CoG, one can calculate the motions of any point on the vessel using superposition(Journée & Pinkster 2001).

The absolute motions are the motions of the vessel in the steadily translating coordinate system, $O(x,\ y,\ z)$. For linearization, the angles of rotation φ, θ and ψ are assumed to be small (i.e. < 0.1 radians). Journée and Pinkster (2001) noted that, the angles must be expressed in radians, because in the linearization, it is assumed that:

$$\sin\varphi \approx \varphi \quad \text{and} \quad \cos\varphi \approx 1.0 \tag{2.1}$$

For the assumed small angle motions, the transformation matrix from the body bound coordinate system to the steadily translating coordinate system is noted by Journée and Pinkster (2001) as:

$$\begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} 1 & -\psi & \theta \\ \psi & 1 & -\varphi \\ -\theta & \varphi & 1 \end{pmatrix} \cdot \begin{pmatrix} x_b \\ y_b \\ z_b \end{pmatrix} = R \tag{2.2}$$

Using the matrix noted by Van der Molen (2010), the transformation of the coordinates of an arbitrary point $P(x_b,\ y_b, z_b)$ on the vessel to an earth bound coordinate system $S(x_p,\ y_p, z_p)$ is:

$$\overline{x}_P = \overline{x} + R \cdot \overline{x}_b \tag{2.3}$$

$$\therefore \begin{pmatrix} \mathrm{x_P} \\ \mathrm{y_P} \\ \mathrm{z_P} \end{pmatrix} = \begin{pmatrix} \mathrm{x} \\ \mathrm{y} \\ \mathrm{z} \end{pmatrix} + \begin{pmatrix} 1 & -\psi & \theta \\ \psi & 1 & -\varphi \\ -\theta & \varphi & 1 \end{pmatrix} \cdot \begin{pmatrix} \mathrm{x_b} \\ \mathrm{y_b} \\ \mathrm{z_b} \end{pmatrix} \tag{2.4}$$

Where $\mathrm{x},\ \mathrm{y}$ and z are the vessel motions surge, sway and heave respectively, while $\mathrm{x_b},\ \mathrm{y_b}$ and $\mathrm{z_b}$ are the body bound coordinates of the arbitrary point on the vessel.

From equation (2.4), the motion components of the arbitrary point, is then given by:

$$\begin{aligned} x_P &= x - y_b\psi + z_b\theta \\ y_P &= y + x_b\psi - z_b\varphi \\ z_P &= z - x_b\theta + y_b\varphi \end{aligned} \tag{2.5}$$

From equation (2.5), it is evident that all the motions have more than one contributing factor. The vertical motion, z_P in point $P(x_b,\ y_b, z_b)$ on the vessel is an example. The vertical motion, z_P is a combination of heave, roll and pitch motions.

2.2.4 Effect of long waves on moored vessels

When long waves enter a harbour primarily designed to reduce short wave energy, they could become trapped inside the harbour basin. The shape and size of such a harbour basin could lead to wave resonance for wave periods that are equal to that of the basin resonance period.

The vertical motions of moored vessels typically have resonance periods of 10 to 15 seconds and are therefore most often induced by short waves at a berth (PIANC 2012). However, another mechanism which drives vertical motions, occurs when a vessel is moored on an anti-node of a standing wave as shown in Figure 2.4. The orbital velocities at an anti-node for a standing wave are vertical, causing heave motions with a similar period to that of the standing wave, which is normally in the long period range in ports. The "slow" vertical movements resulting from the standing wave would therefore be far from the vertical resonance period of the vessel, and unlikely to cause a problem (Stuart 2013).

For a moored vessel however, horizontal motions typically resonate at longer periods which range from 30 to 300 seconds, depending on vessel size, type and mooring set-up (PIANC 2012). The surge motions will be more severe when a vessel is placed on a node as shown in Figure 2.4. Nodes are characterized by horizontal oscillating velocities and no change in the surface elevation. In addition to these horizontal currents, the slope (angle) of the water surface may also induce surge motions. The resulting force on the mooring system, induced by the water surface slope, may be described as:

$$F = \sin(\theta) \times M \times g \tag{2.6}$$

Where θ is the angle of the water surface to the horizontal, M is the ship mass and g is the acceleration of gravity (Stuart 2013).

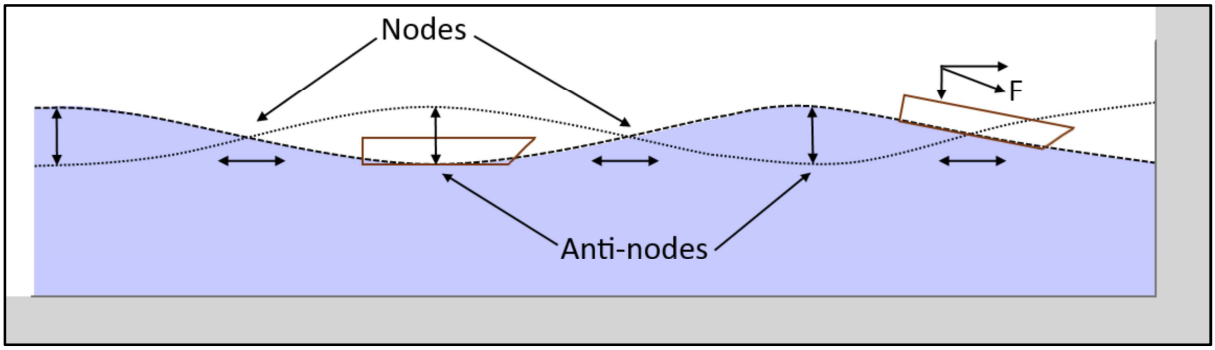

Figure 2.4: Ship motions in standing wave (Stuart 2013)

The resonance periods of the moored vessel are defined by the combination of vessel size and mooring system; in general a stiffer mooring system results in higher response frequencies (PIANC 2012). Stiffening the mooring system in order to decrease the resonance periods below the long wave periods in the port is often the first action taken to mitigate vessel motions. Large vessels typically have a lower natural frequency compared to smaller vessels (PIANC 2012).

For the (un)loading of container vessels, surge motion is identified as the critical criteria for large container vessels which are not exposed to broad-side waves. When surge motion for a large container vessel is within acceptable limits, the other motion components are generally also accepted as within limits. Table 2.1 provides information on the maximum allowable surge motion to achieve a (un)loading efficiency of 95% for large vessels (PIANC 2012).

Table 2.1: Maximum significant surge motion amplitudes for an (un)loading efficiency of 95%, for large container vessels (PIANC 2012)

Container placing criterion	Basis for placing criterion	Maximum allowable significant surge motion amplitude (T_{surge} = 30 s - 100 s)
0.1 m	Twist-lock pins	0.2 m
0.2 m	Spreader flaps	0.4 m

Vessels smaller than Panamax size, also referred to as small vessels in this thesis, however often become the critical vessel for mooring design. This is due to smaller vessels typically having a higher natural frequency to that of large vessels. For smaller vessels, each degree of motion is seen as critical for (un)loading. Table 2.2 provides the maximum allowable motion criteria of all 6DOF to achieve a (un)loading efficiency of 95% for small vessels (PIANC 2012).

Table 2.2: Maximum allowable significant motion amplitudes for an (un)loading efficiency of 95%, for smaller container vessels (PIANC 2012)

Principal motion	Maximum allowable significant motion amplitude
Surge	0.2 m to 0.4 m
Sway	0.4 m
Heave	0.3 m
Roll	1.0°
Pitch	0.3°
Yaw	0.3°

These maximum allowable motions presented in Table 2.1 and Table 2.2 is normally not where line breaking will occur. Infragravity or long waves which are parallel to the quay where vessels are moored, normally has a greater effect on the surge motion of larger vessels. This is due to long waves having wave periods which are close to the typical natural surge frequency of large vessels which cause a build-up in surge motion. Line breaking almost always occurs due to large surge or sway motions. As a conservative rule of thumb, line breaking can start at movement which exceeds three times the maximum significant motion used for an (un)loading efficiency of 95%. For surge and sway, depending on the mooring arrangement, pre-tensioning of mooring lines and mooring line age, this can start to occur when motions exceed 1.2 m.

2.3 Prototype Measurements of Vessel Motions

Monitoring of motions is nowadays common on board most ships, drilling rigs and their offshore support vessels. Some of the motion measurement instruments use three-axis accelerometers, solid state gyros or even fiber Optic Gyros. These sensors provide continuous and accurate records of motions in 6DOF and are compact in size. Illustrations of how an accelerometer and gyroscope work are presented in Figure 2.5 and Figure 2.6 respectively.

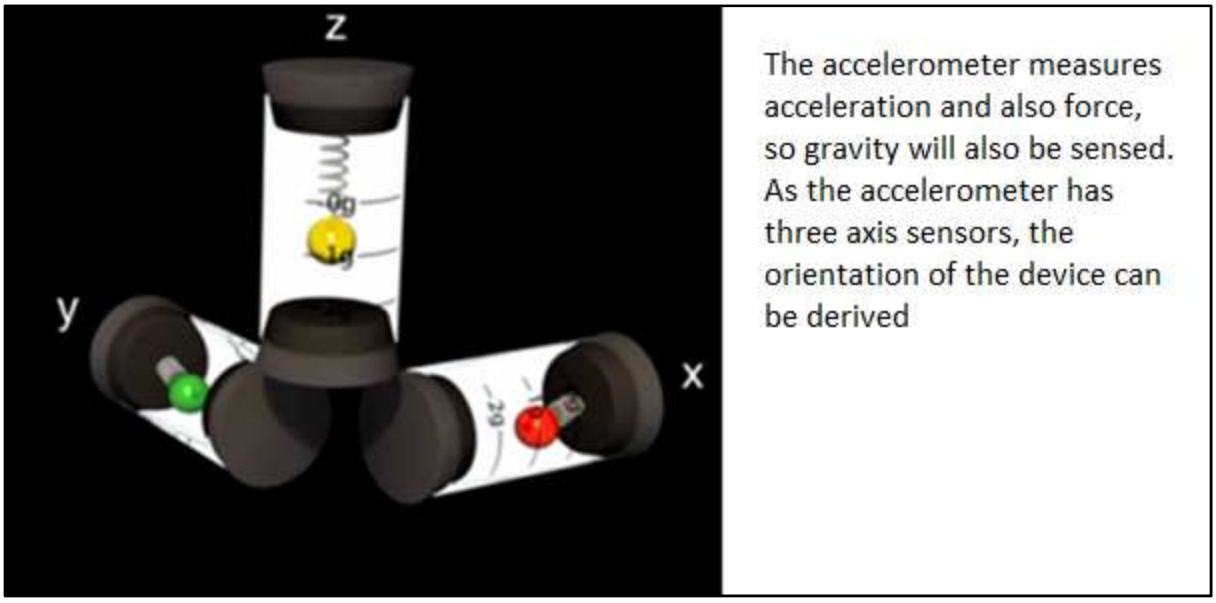

Figure 2.5: Visual illustration of how an accelerometer work (Liang 2013)

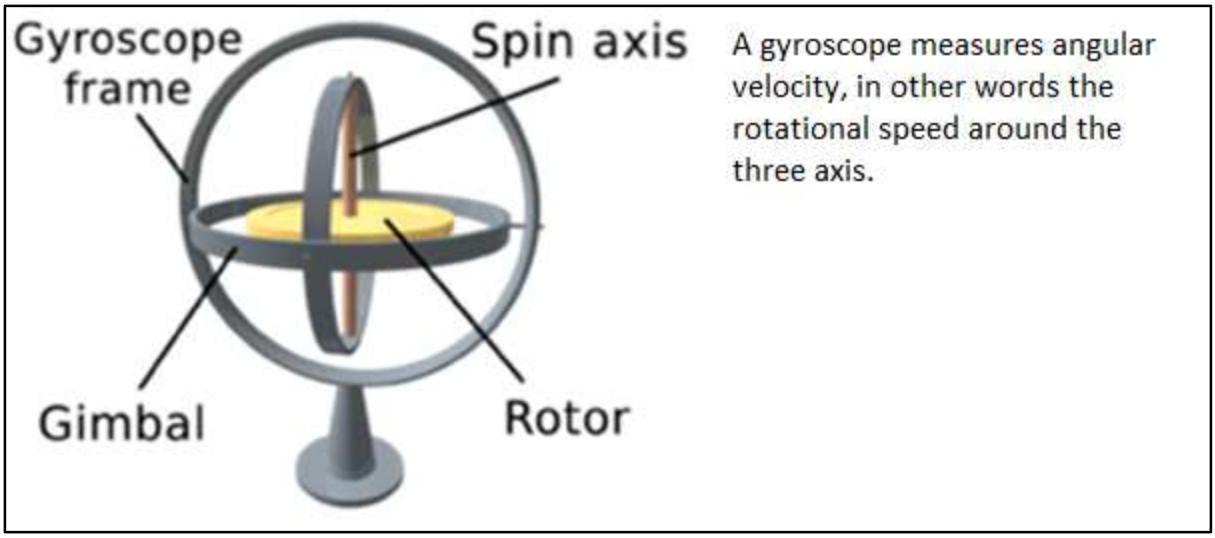

Figure 2.6: Visual illustration of how an gyroscope work (Liang 2013)

Motion recording data which includes motions, accelerations or velocities at specific locations on a ship is invaluable since it can be utilized to better design the operational criteria of a ship, drilling rig and offshore support vessel (MARIN n.d.).

Standard 6DOF sensors can measure vessel motions with periods up to 20 or 30 seconds. For longer periods, more sophisticated instrumentation like Differential Global Positioning Systems (DGPS's) are necessary, but only in 2 or 3 degrees of freedom (MARIN n.d.), when used on their own. Standard 6DOF sensors are not sensitive enough to accurately measure changes for low frequency motions.

The subsections below highlight current methods used in the prototype (real situation as opposed to model situation) to accurately measure long period waves of moored vessels in a harbour.

2.3.1 DGPS system

MARIN (n.d.), states that when DGPS's are used to measure low frequency motions due to long waves, only measurements in 2 or 3 degrees of freedom are possible. However, by using ship particulars and other loading information, it becomes possible to obtain all 6DOF by post processing of recorded data.

The CSIR used a DGPS to measure long wave induced movement of moored ships at the Port of Ngqura (CSIR et al. 2013). The equipment only measures movement and displacement at a particular point. To obtain vessel motions, the movement of each individual point is translated to vessel coordinates. The vessel information and loading conditions of the ship are used to determine the CoG position around which translation and rotation is calculated. Figure 2.7 illustrates the typical positions of GPS devices on a vessel to obtain the desired motions. A base station, sending corrections, is also erected alongside the quay or piled dock.

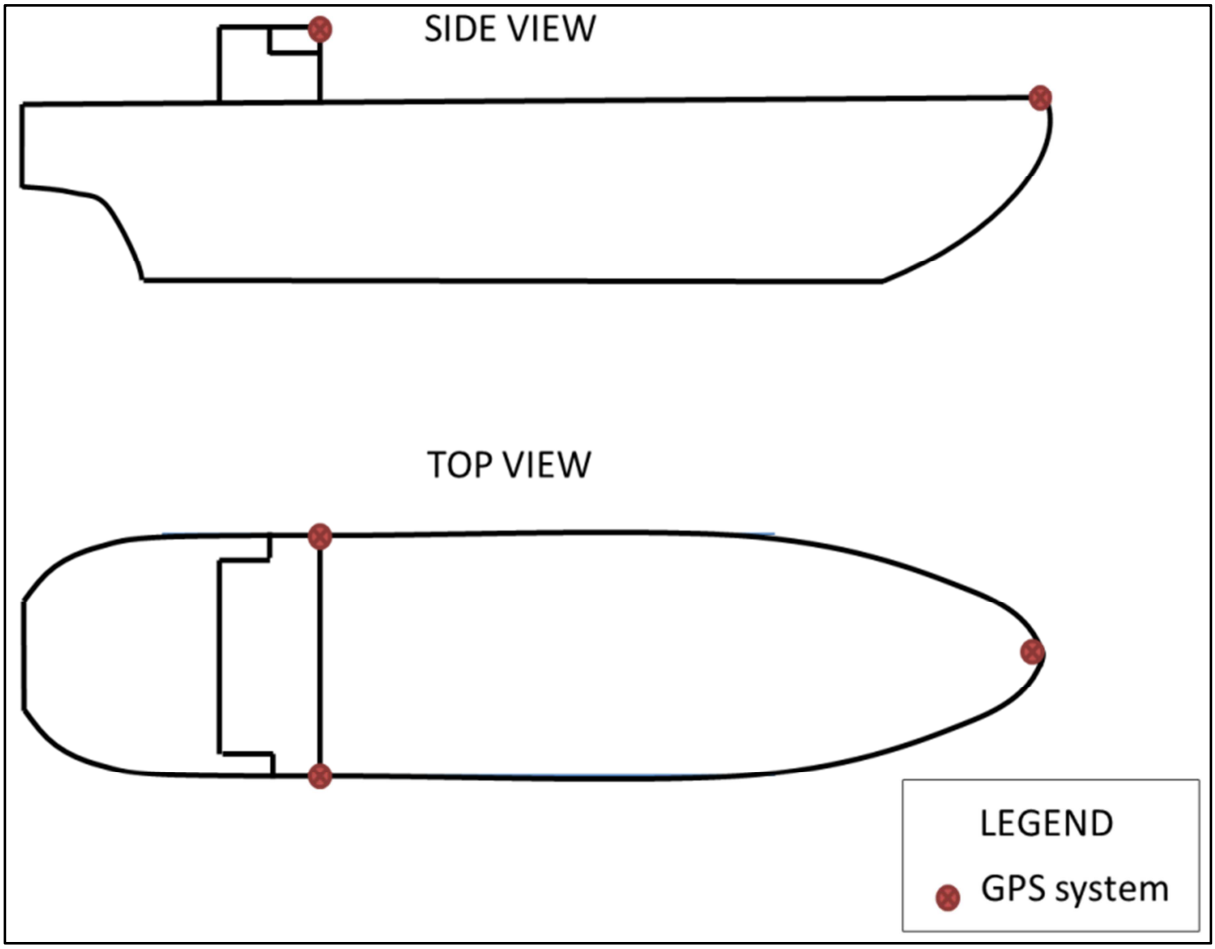

Figure 2.7: Typical GPS locations

2.3.2 Video tracking system

A pilot project was conducted at the Port of Saldanha in 2008 (van Son 2008). Collaborative research between the CSIR and Enviro Vision Solutions (EVS) produced a system to monitor the motions of moored iron ore carriers at the loading jetty.

The pilot system assumed a rigid body vessel while points near the ends and sides of the vessel (bow, stern, port and starboard side) were tracked with a camera facing the vessel at an oblique angle. Basic vessel dimensions and the location of each point being tracked relative to the camera, jetty and vessel coordinate systems were input parameters to the tracking system, as illustrated in Figure 2.8.

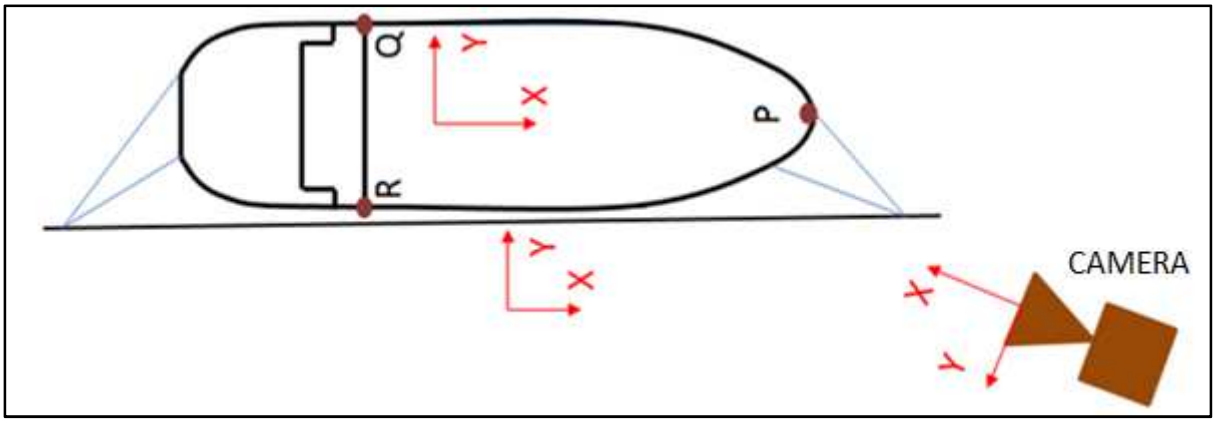

Figure 2.8: Illustration of different coordinate systems used in van Son (2008)

Software developed by EVS tracks horizontal and vertical pixel movement of three points on the vessel (P, Q and R). Pixel displacements are converted from the two dimensional (2D) camera plane to 3D object motions, by using cosine projection and by knowing the coordinates of the points being tracked as well as the view angle and coordinates of the camera. Algorithms convert pixel displacements from the camera coordinate system to the jetty coordinate system and finally to the vessel coordinate system. Motions in pixel units are finally converted to vessel motions in metre, by knowing the physical dimensions of the ship

The tracking windows within the software developed during the project are shown in Figure 2.9 and below.

Figure 2.9: Tracking windows arround targets (van Son 2008)

2.4 Physical Modelling and Vessel Motion Measurements

Small-scale physical models of vessels are an invaluable tool to test the behaviour of moored ships in various wave conditions and design layouts. Small scale testing are mainly done to optimize port or structure designs for various wave conditions before constructing the chosen design layout in prototype.

The Recommendations to the 22nd International Towing Tank Conference (ITTC) state that the measurement of 3D vessel motions was traditionally done with potentiometer systems attached to the model, or with accelerometers and gyroscopes (ITTC 1999).

Presently, optical (i.e., image-based) motion measurements of vessel models have become the standard in most model basins (ITTC 2008). The subsections below highlight some of the physical model measurement techniques currently used to measure vessel motions in hydraulic laboratories.

2.4.1 MOTAN system using accelerometers and angular rate sensors

The Canadian Hydraulics Centre (CHC) uses a system developed in-house, namely MOTAN, which is an acronym for MOTion ANalysis system. The inertial motion sensor unit of MOTAN uses three linear servo accelerometers and three micro-machined quartz angular rate sensors. The accelerometers measure the total acceleration along X, Y, and Z body axes of the ship model, while the angular rate sensors measure angular rotation vectors of the model vessel (Briggs & Melito 2008). Figure 2.10 shows a typical installation of the MOTAN. The fixing of the measurement instrument on to the vessel makes this method intrusive.

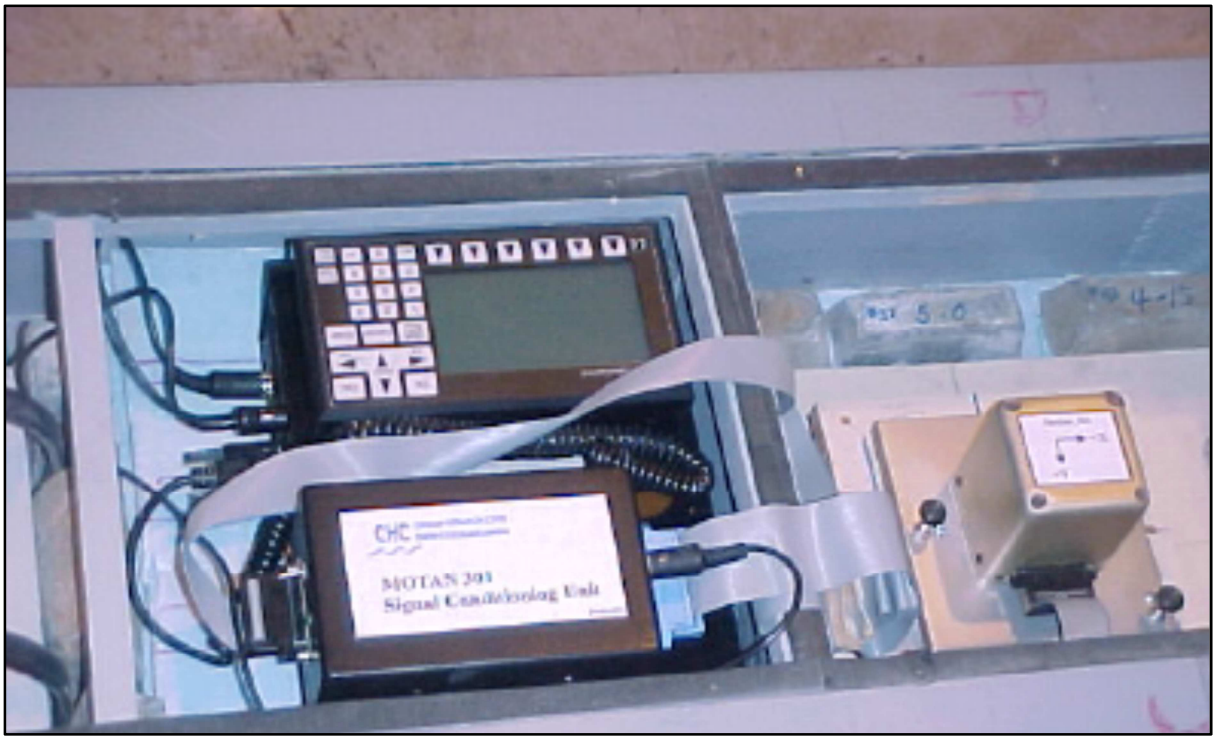

Figure 2.10: Typical installation for MOTAN motion analysis system (Briggs & Melito 2008)

2.4.2 Qualisys system using active infrared targets

The Hydraulics Laboratory at the University of Porto in Portugal measure moored vessel motions by using a system from Qualisys in Sweden.

The Qualisys system uses three infrared cameras that emit infrared light. Reflective sphere markers are placed on top of the vessel to allow easy detection by cameras. Vessel movements are recorded to a PC through the duration of a model test. The recorded footage is synchronised through the same cable connection and is post processed after the test to graphically present the animated motions of the reflective markers. The position of the markers through time is then translated to a mathematical model of the ship which allows the calculation of all 6DOF for the vessel (Malheiros et al. 2009). Figure 2.11 shows reflective sphere markers on a model vessel and Figure 2.12 presents the animated motions graphically. This method is intrusive due to the reflective spheres which have to be fixed on the vessel.

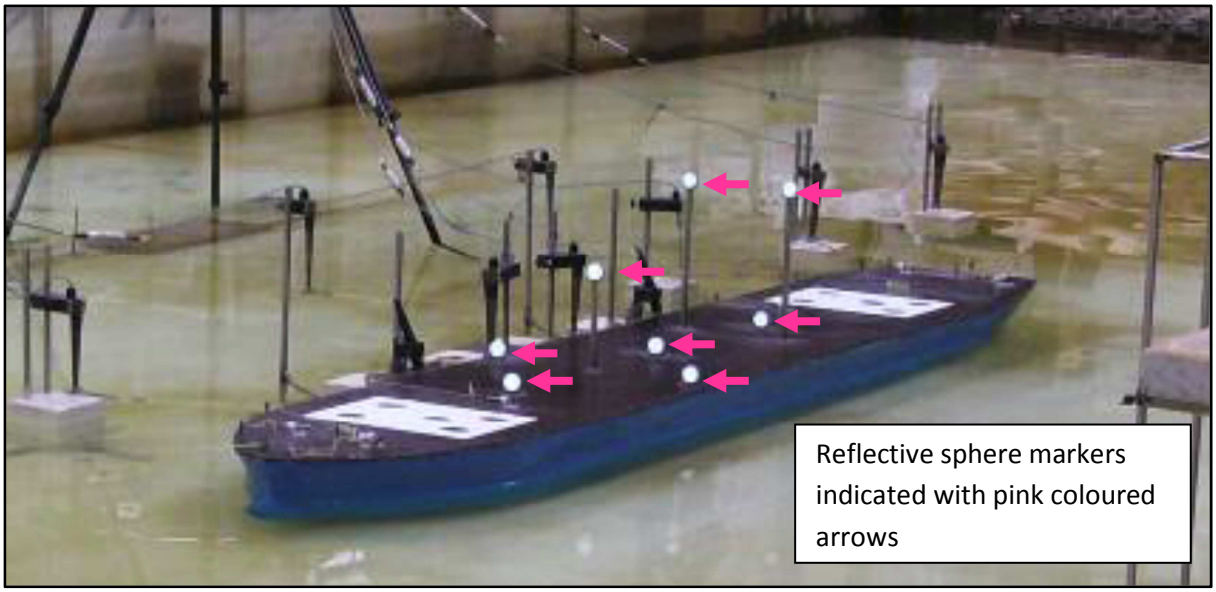

Figure 2.11: Reflective sphere markers for Qualisys system (Malheiros et al. 2009)

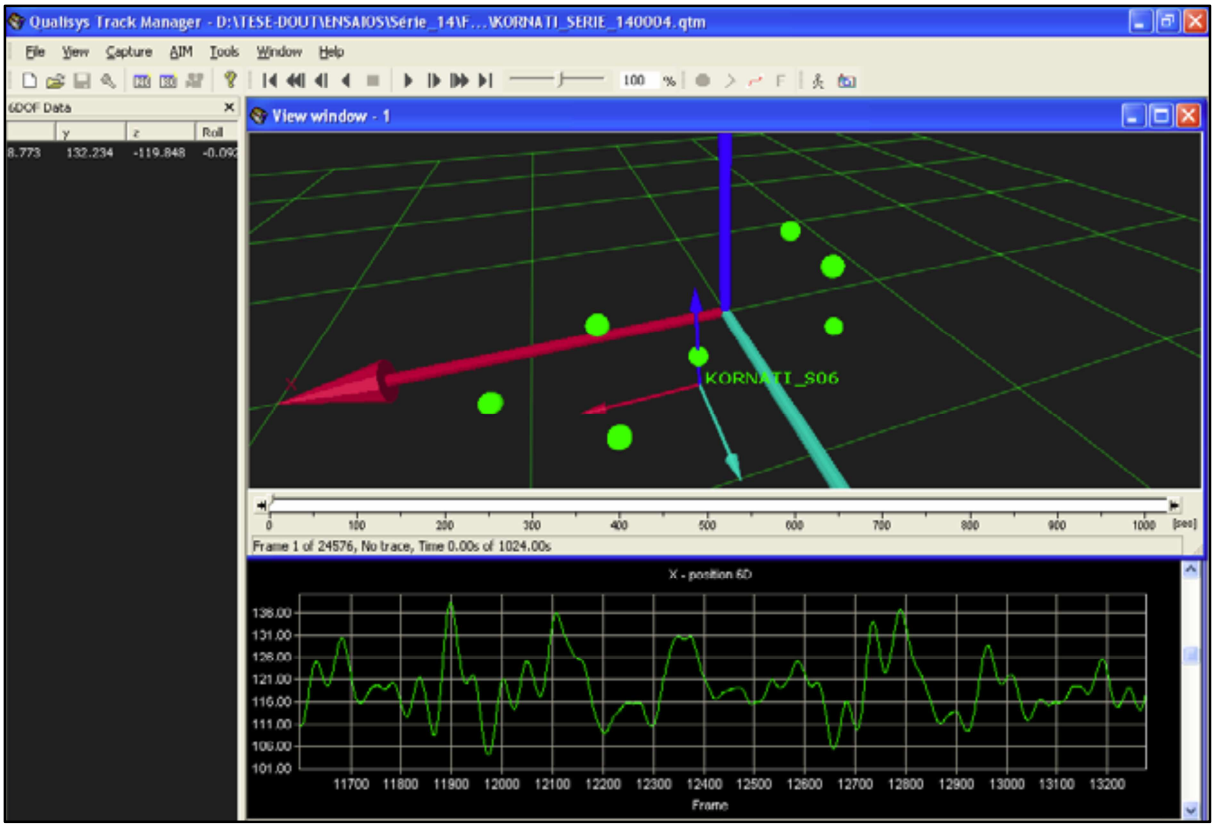

Figure 2.12: Screenshot showing graphical presentation of Qualysis markers (Malheiros et al. 2009)

2.4.3 Imaged based system using single flat known target

An imaged-based method proposed and tested by Benetazzo (2011) at the Experimental Center for Hydraulic Modelling in Padua, Italy. The optical method presented in Benetazzo (2011) is based on a single camera view of a known flat chessboard target. This target is fixed to the floating structure of which 3D motion measurements are required. The 6DOF motions (surge, sway, heave, roll, pitch and yaw) are then calculated by using known square sizes as a reference system defined for such a panel (Benetazzo 2011). An example of a flat chessboard-like panel on top of a small-scale floating object is shown in Figure 2.13. The measurement plate which needs to be mounted on the floating object makes this system intrusive.

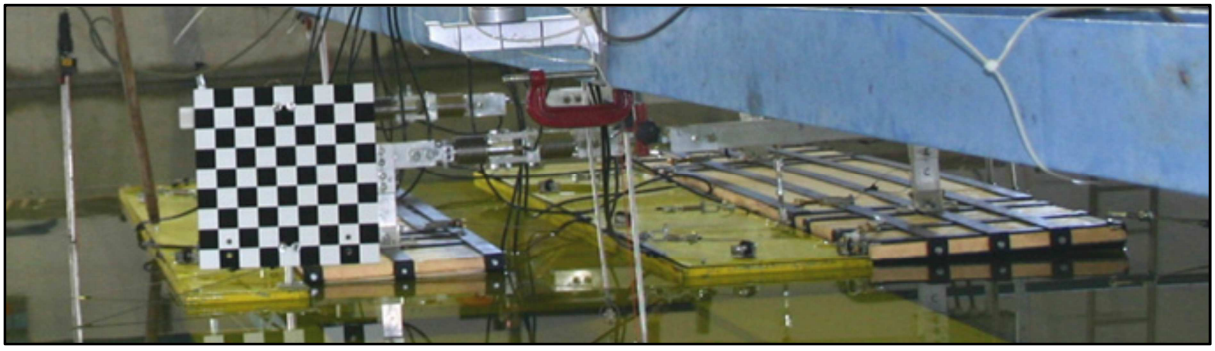

Figure 2.13: Example of chessboard-like panel on small-scale floating body (Benetazzo 2011)

2.4.4 Keoship optical tracking system

The Hydraulics Laboratory at the CSIR, in Stellenbosch developed a moored vessel motion monitoring system called the Keoship system (Van der Molen et al. 2010). Two light metal plates are placed on the deck of the model ship together with two mirrors placed at a 45° angle above these plates. Two video cameras are then placed alongside the ship, focusing on the side and the reflected top view of each plate. Figure 2.14 shows a typical set-up of a Keoship measurement system while Figure 2.15 shows a typical camera perspective view.

Figure 2.14: Typical Keoship set-up at the CSIR

The cameras are used to track the pixel displacements of the black and white interface stickers on each of the plates. The pixel displacements are converted to horizontal and vertical distance displacements by using the calibration strips also in view of the cameras.

The location of these markers in vessel coordinates is used to translate the vertical and horizontal displacements to 6DOF vessel motions.

The vessel motions, along with mooring information, fairlead locations, ship particulars and loading information are then used to translate vessel motions to reaction forces in each mooring line and on each fender due to vessel motions. However, the measurement plates that have to be put on the ship, make this system intrusive.

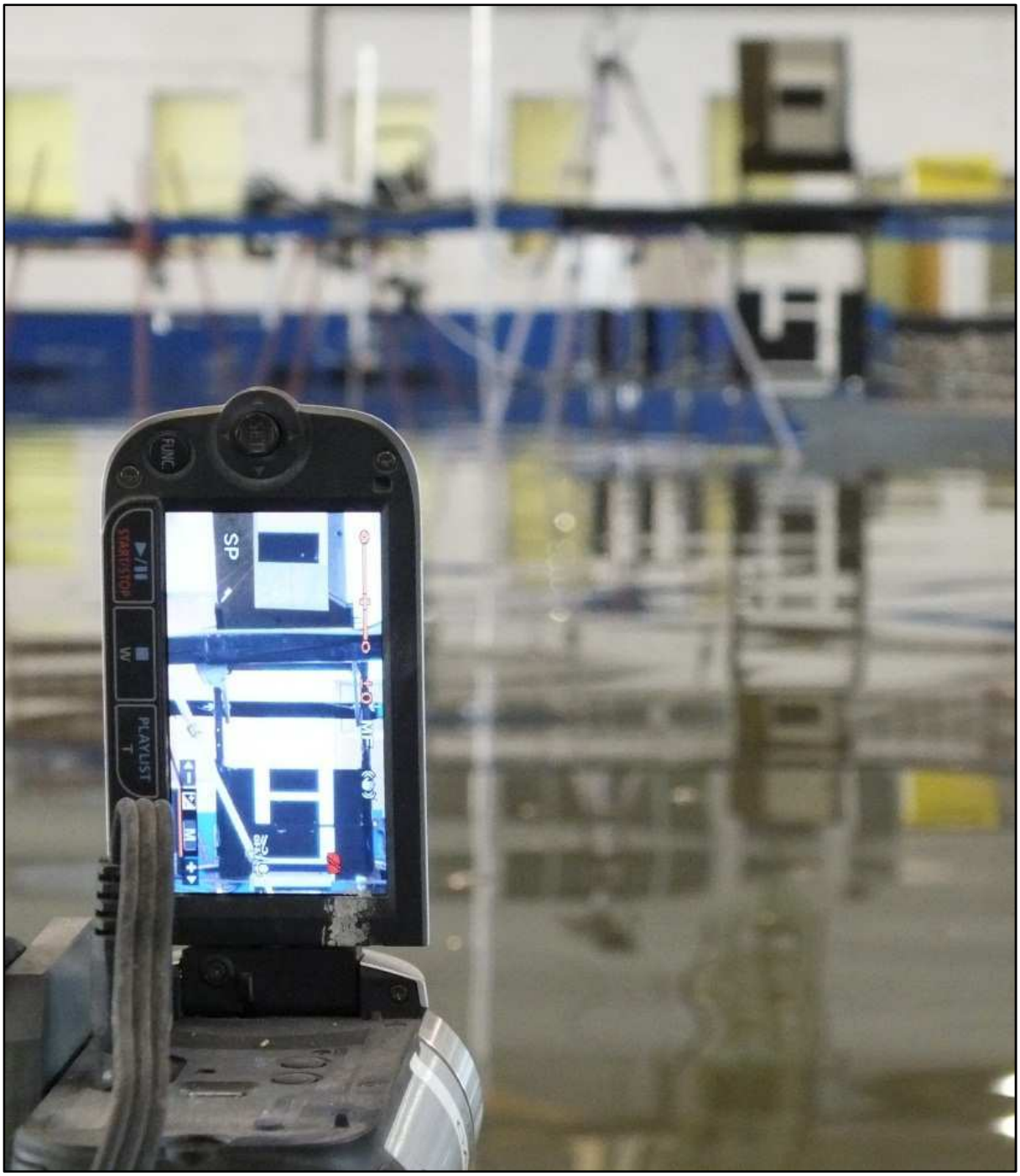

Figure 2.15: Typical Keoship camera view perpendicular to the bow

2.4.5 Strain gauge force system

The Hydraulics Laboratory at the CSIR in Stellenbosch, also uses strain gauges to measure mooring line and fender forces which are then converted to moored vessel motions.

The strain gauge method uses mooring lines and fenders which are calibrated to have equivalent stiffness characteristics to that of the prototype counterpart. The forces acting on these mooring lines and fenders are then measured with the strain gauges.

The forces, along with fender contact locations, fairlead locations, ship particulars and loading information, are then used to translate the measured forces to 6DOF vessel motions. This is also an intrusive system.

An illustration of mooring lines being represented by springs and string is shown in Figure 2.16, while a typical model set-up of mooring lines connected to strain gauges are shown in Figure 2.17.

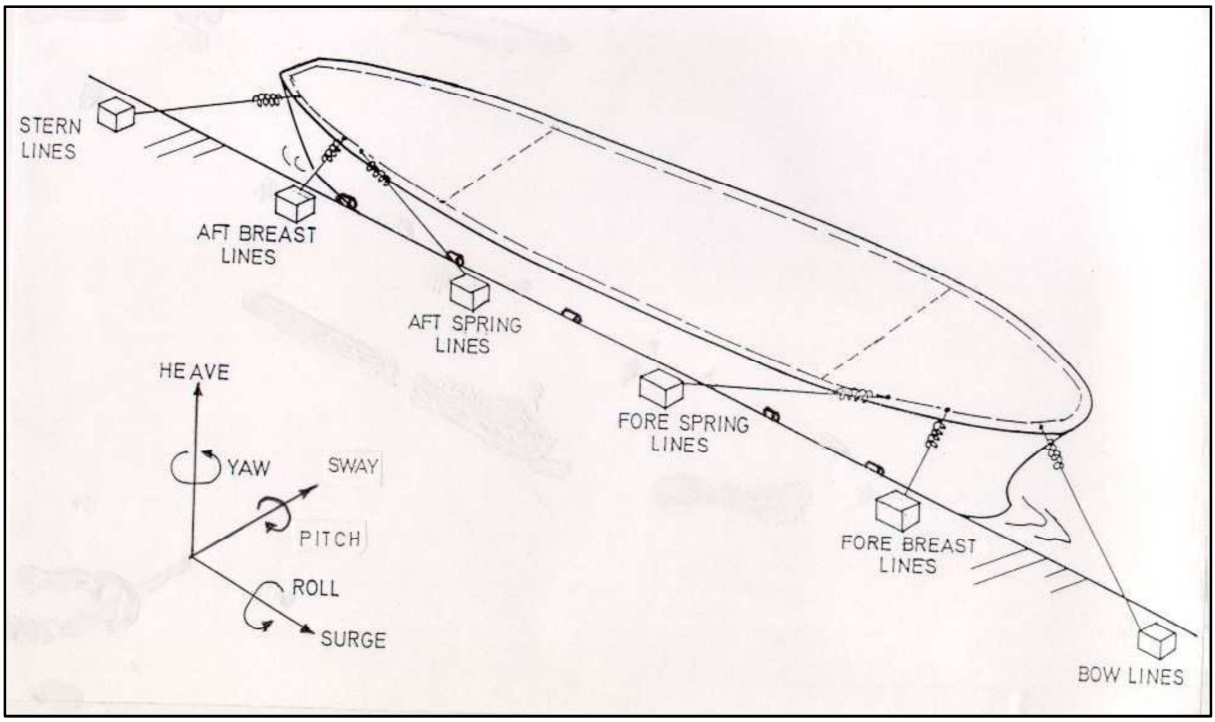

Figure 2.16: Illustration of mooring lines represented by springs and string (Van der Molen & CSIR 2010)

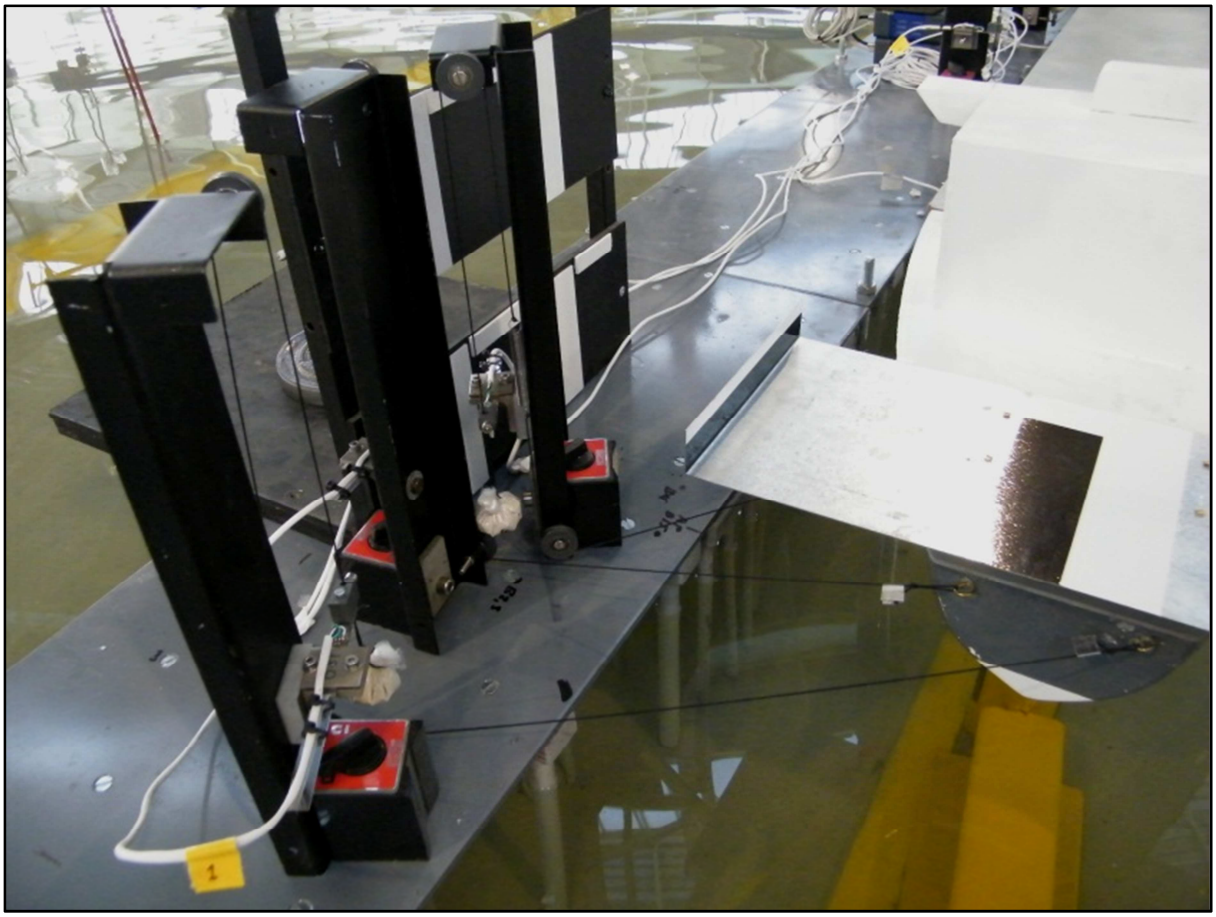

Figure 2.17: Typical mooring line set-up at the CSIR using strain gauges (CSIR et al. 2008)

2.5 Non-intrusive Object Tracking

Section 2.4 shows that current video monitoring tracking systems used for measuring the movement of a small scale vessel while berthed uses plates, targets or accelerometers which are mounted onto the vessel. These methods are all intrusive in nature, and in addition to the mounting of plates, targets or sensors, require mounting coordinates to translate measured displacements to 6DOF vessel motions. Although Benetazzo (2011), claims his imaged-based method as briefly presented here in section 2.4.3, is non-intrusive, the method uses a checker board plate mounted on a vessel and is therefore intrusive.

According to the literature review done, there is currently no non-intrusive method being used or researched for measuring movement or motions of a moored vessel. Current non-intrusive object tracking methods are mainly used in the Industrial sector (i.e. to position and assemble parts in a motor vehicle assembly line). This section summarises the literature review into non-intrusive object tracking.

A typical task in image processing for computer vision applications is to identify specific objects in an image and then to determine each object's position and orientation relative to some coordinate system. An example of how the information can be used, is in the manufacturing industry, where robotic equipment manipulates the position and orientation of objects in assembly lines. The combination of position and orientation is referred to as the “pose” of an object (DigitalRune n.d.).

A robust model-based visual tracking algorithm that gives accurate 3D pose for a rigid object is presented in Yoon et al. (2005). Similar to other typical model-based vision systems, the pose estimation algorithm used by Yoon et al. (2005) projects the model edges of the object being tracked into a scene image in order to find the best matching scene edge for a given model edge. The pose of the target being tracked is then updated before the next image is put into the tracking algorithm.

The model edges referred to above, would all be the straight lines and intersection points within the tracking area for the current image frame being sampled. The scene image referred to above would be a wire frame mesh model of the object being tracked, which includes x, y and z coordinates of all the straight lines and intersection points of the object.

The system presented by Yoon et al. (2005) can estimate accurate pose by matching only a small number of features. Yoon et al. (2005) used a 3D range data of the target object. This was acquired by using a 3D laser scanner. An example picture of the rendered range data image of a target object along with its corresponding wire frame model, is given in Figure 2.18.

Figure 2.19 shows the flow diagram of the pose estimation algorithm presented in Yoon et al. (2005). The flow diagram shows how the algorithm iteratively goes through processes to obtain the final pose of an image for a set of scene features.

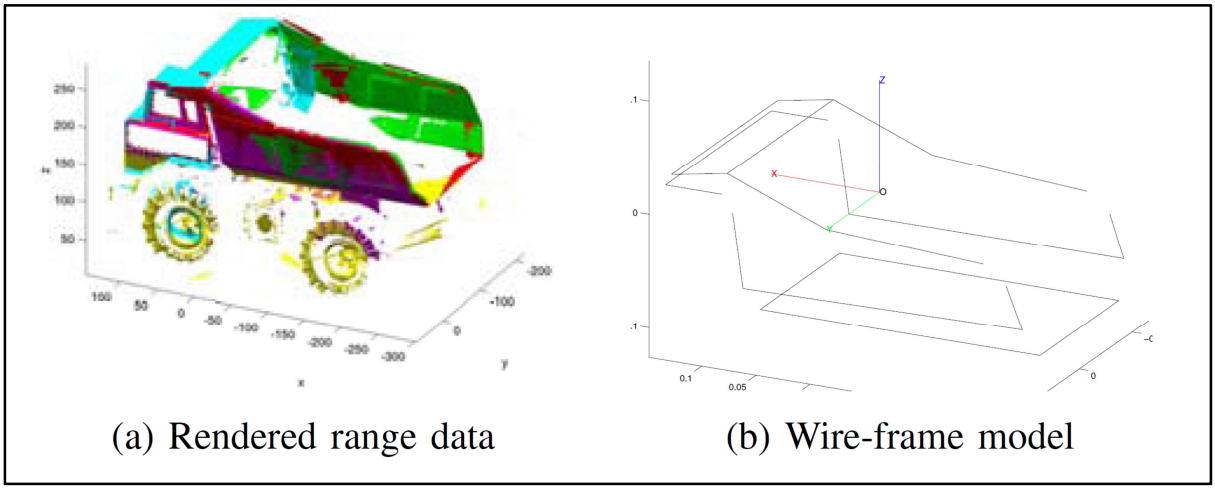

Figure 2.18: Model of a target object being tracked (Yoon et al. 2005)

The pose estimation algorithm presented in Yoon et al. (2005) is taken even further in Yoon et al. (2008). The updated pose estimation algorithm more accurately tracks large and jerky motions. Both tracking algorithms use an additional filter algorithm. The filter algorithm uses a series of measurements, containing noise and other inaccuracies, to produce estimates of the unknown variables used to update the pose mean and covariance of the object between images. The filter algorithm used, is also known as the Extended Kalman filter.

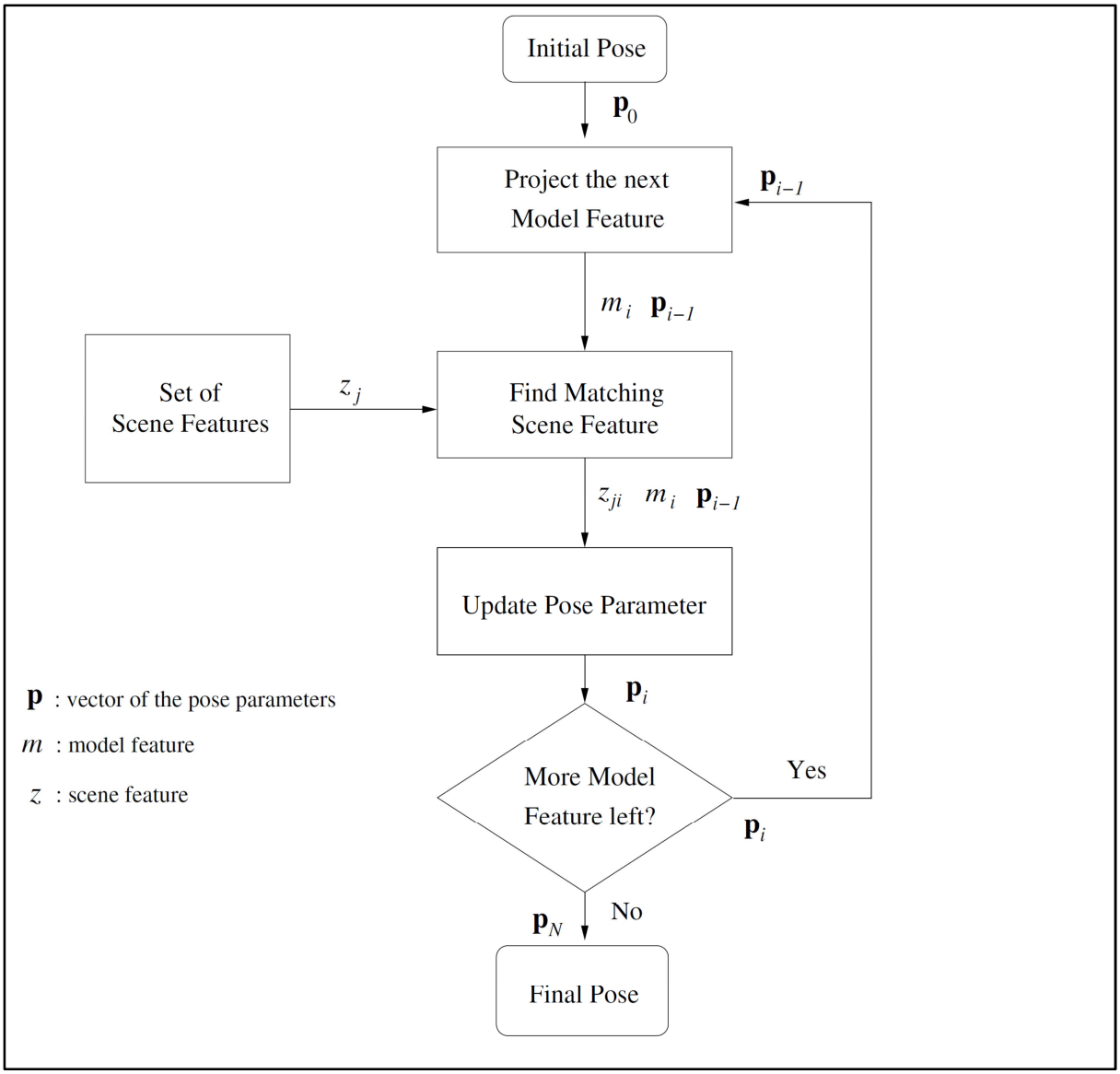

Figure 2.19: Pose estimation algorithm overview (Yoon et al. 2005)

The 3D pose is updated between image frames. The uncertainty associated with such updates becomes inappropriate when model features are matched to scene features for large and jerky motions (Yoon et al. 2008).

Yoon et al. (2008) re-examines the model to include scene feature pairings after each pose update. If it is found that the new pose does not support the pairings, the estimated poses are undone and new pairings sought. The process is repeated until the updated pose and the set of matched model to scene features support each other completely.

Yoon et al. (2008) state that, 'object tracking has numerous applications such as traffic surveillance, augmented reality, mobile robot navigation, robotic assembly on a moving line, etc. For many of these applications involving 3D objects, it is not sufficient to just do 2D tracking; the tracking algorithm must also provide the 3D pose of the object.'

Since the 3D pose of a rigid object involves 6 DOF (three translations and three rotations), the tracking algorithm for any of the earlier mentioned applications must yield all six pose parameters (Yoon et al. 2008). Any obstructions, background clutter and varying illumination conditions should

also be taken into account while estimating all of these six parameters. An overview of the visual tracking algorithm used by Yoon et al. (2008) is shown in Figure 2.20. The flow diagram shows how the algorithm iteratively goes through pose estimation module and pose prediction module to obtain the final estimated pose of an image for a set of scene features.

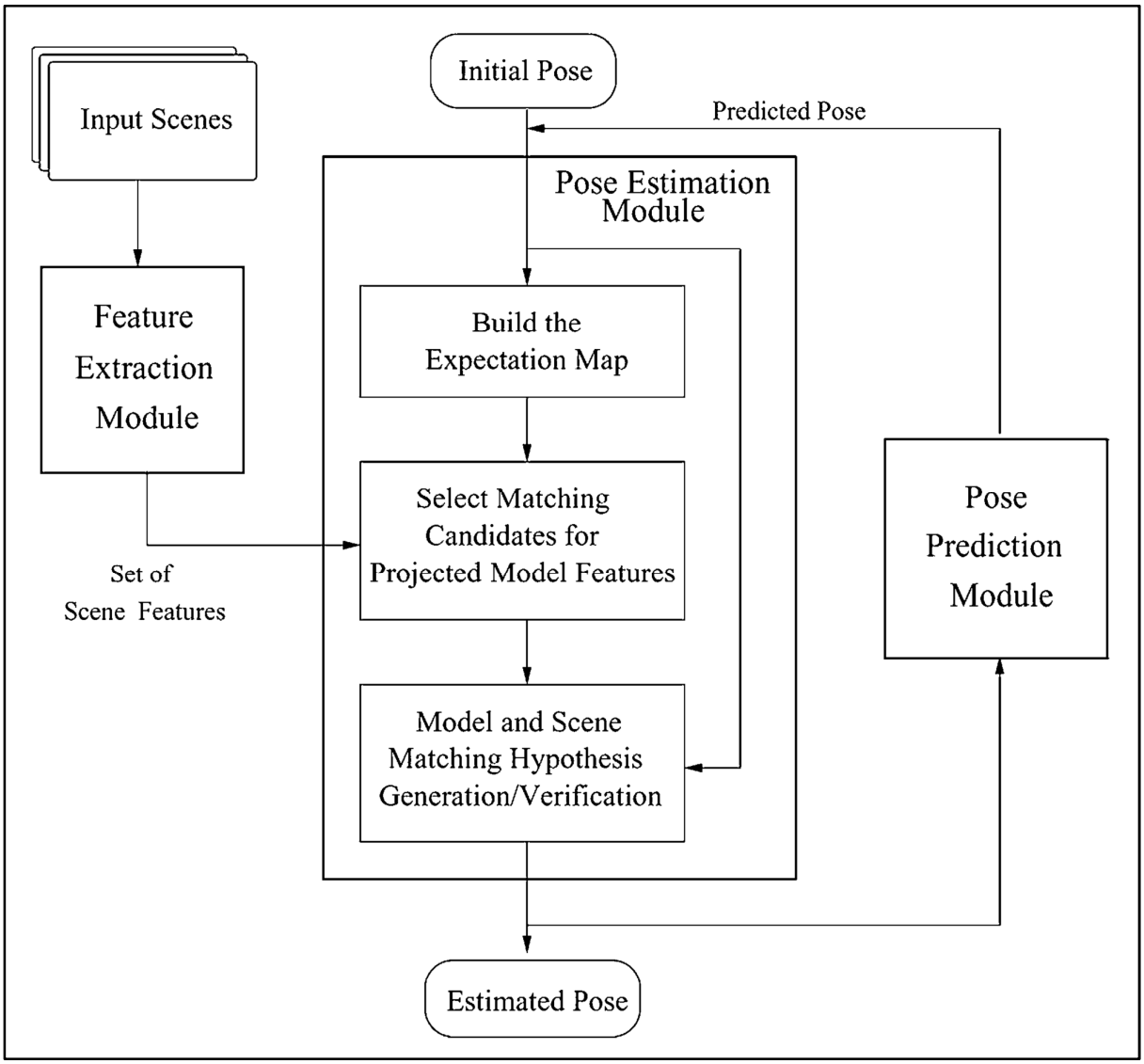

Figure 2.20: Pose estimation algorithm overview (Yoon et al. 2008)

2.6 Summary

The literature review indicated that there is currently no non-intrusive method being used or researched for measuring vessel motions of a moored vessel for physical model use as well as prototype use in the field. The experiment plan which follows, will include a novel approach not yet implemented for physical model use or in the field to measure vessel motion by tracking a 3D object on a vessel.

3 EXPERIMENT PLAN

3.1 Verification and Accuracy Tests

In order to determine the feasibility of the tracking algorithm developed, a few 'static' verification tests will be conducted. The verification tests involve manually moving or rotating the object being tracked, a known distance or angle and verifying the tracking displacement obtained with values physically measured.

The verification tests plan to include a complete system assembly, which will be used in the physical model testing. For verification of the tracking algorithm, only motions (surge, sway and yaw) in the horizontal plane will be looked at from three different view angles (the first being perpendicular to the surge direction, the second being perpendicular to sway and the third view angle being oblique to both the surge and sway direction).

Once the tracking algorithm is verified, accuracy tests will be done for all 6DOF motions. The accuracy tests will be done over a range of displacements for both translation and rotation. The results would assist in determining if the percentage error over a range of translation and rotation distances is constant, linear, decreasing or increasing.

3.2 Physical Model Testing

Plans for the physical model testing in the research study contain two experimental measurement methods for moored ship motions. The methods will include a 2D Light Emitting Diode (LED) tracking algorithm and a 3D Object tracking algorithm. The physical model will be set up at a scale of 1:100 in the Hydraulics Laboratory of the CSIR in Stellenbosch. Both methods will be developed by the author and will include some programming assistance from the CSIR.

The first experimental set-up will compare movement results obtained from the proof-of-concept 2D LED tracking algorithm to that which is obtained from the Keoship and force measurement systems.

The second experimental set-up will compare results obtained from the proof-of-concept 3D Object tracking algorithm to that obtained from the Keoship and force measurement systems.

The Keoship and force measurement methods are currently being used in the Hydraulics Laboratory of the CSIR. Detail on the accuracy of the Keoship system can be found in Moes and Hough (1999). The experiments will use video, model and data equipment of the CSIR.

Comparison of the data sets will include validation for all 6DOF motions. It will also verify the feasibility of the methods for each of the respected vessel motions. The proposed methods and test plans are presented in the subsections below.

The test plans incorporate two different wave heights and periods which are similar to that used for an actual physical model study recently done at the CSIR. A wave height of 1.5 m and 3.0 m in prototype along with wave periods of 12s and 16s were used. The selection of wave conditions will make it possible to establish the viable application limits of each individual method for typical wave conditions used in a vessel motion study. The wave condition range will cover both "small" and "large" motions which range from "smooth" to "jerky".

The testing plans will also include tests with the same input wave conditions to determine repeatability of measurement methods as well as validating the correct working of all the equipment used.

3.2.1 2D LED tracking

This method is to be used as a starting point for development. It is also somewhat intrusive and involves placing light strips of a known shape and size onto the deck. During each test a video camera will be used to capture the movement of the light strips on a data capture card. The video content on the data capture card will then be put into a motion tracking algorithm which is used to analyse the video material. To calculate the 6DOF motions, the tracking algorithm will make use of the known physical dimensions and shape of the LED strips, together with the orientation and location changes of the LED strips between image frames.

All the tests will be conducted under artificial lighting to minimize the effect changing light levels may have on the tracking algorithm. The test plan also includes repeatability tests for each of the conditions.

Figure 3.1 illustrates the proposed placement of a LED rectangle and view angle while the experimental planned tests are illustrated in Table 3.1.

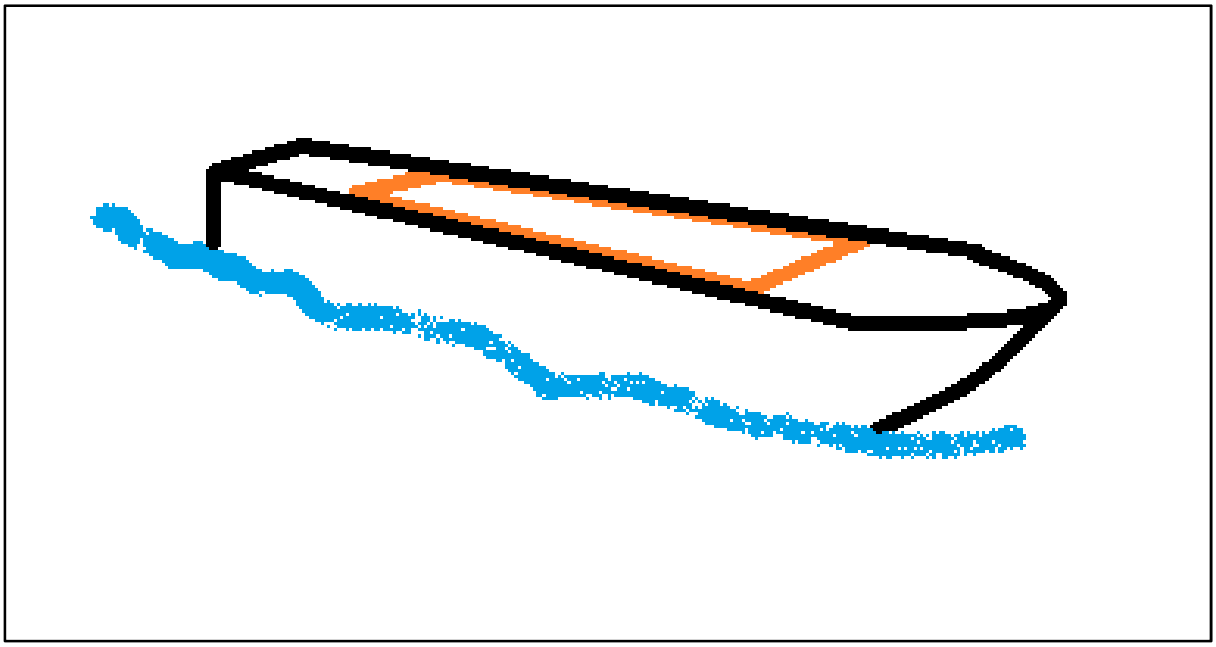

Figure 3.1: Proposed location of LED rectangle on vessel deck indicated with an orange rectangle

Table 3.1: Test plan for 2D LED tracking

Test number	Model		Prototype		Test condition
	Hm0 (m)	Tp (s)	Hm0 (m)	Tp (s)	
LED01	0.015	1.6	1.5	16	Testing tracking algorithm for “small”, ”smooth” waves
LED02	0.015	1.6	1.5	16	Repeatability test for test number 1
LED03	0.015	1.6	1.5	16	Repeatability test for test number 1
LED04	0.030	1.6	3.0	16	Testing tracking algorithm for “large”, ”smooth” waves
LED05	0.030	1.6	3.0	16	Repeatability test for test number 4
LED06	0.030	1.6	3.0	16	Repeatability test for test number 4
LED07	0.015	1.2	1.5	12	Testing tracking algorithm for “small”, ”jerky” waves
LED08	0.015	1.2	1.5	12	Repeatability test for test number 7
LED09	0.015	1.2	1.5	12	Repeatability test for test number 7
LED10	0.030	1.2	3.0	12	Testing tracking algorithm for “large”, ”jerky” waves
LED11	0.030	1.2	3.0	12	Repeatability test for test number 10
LED12	0.030	1.2	3.0	12	Repeatability test for test number 10

3.2.2 3D Object tracking

The second proposed method is slightly more difficult and can be seen as non-intrusive. This involves using a model based algorithm to give accurate 3D pose of the rigid object which will then be translated into vessel motion.

Similarly to the LED tracking method, the video material obtained during tests will be put into to an algorithm which calculates the 6DOF motions. Within the algorithm a commonly used pose estimation approach is however used for the object tracking method. The target object to be tracked will be measured beforehand to create a wire-frame model of the target object. The algorithm then uses the wire-frame model of the target object as the model features to be tracked. The object being tracked will typically be the super structure of a ship and will not include the placement of target objects onto the vessel. However, for model validation purposes the superstructure will be represented by 3D objects with known dimensions.

Figure 3.2 illustrates a typical superstructure to be tracked on the vessel, while the experimental planned tests are illustrated in Table 3.2.

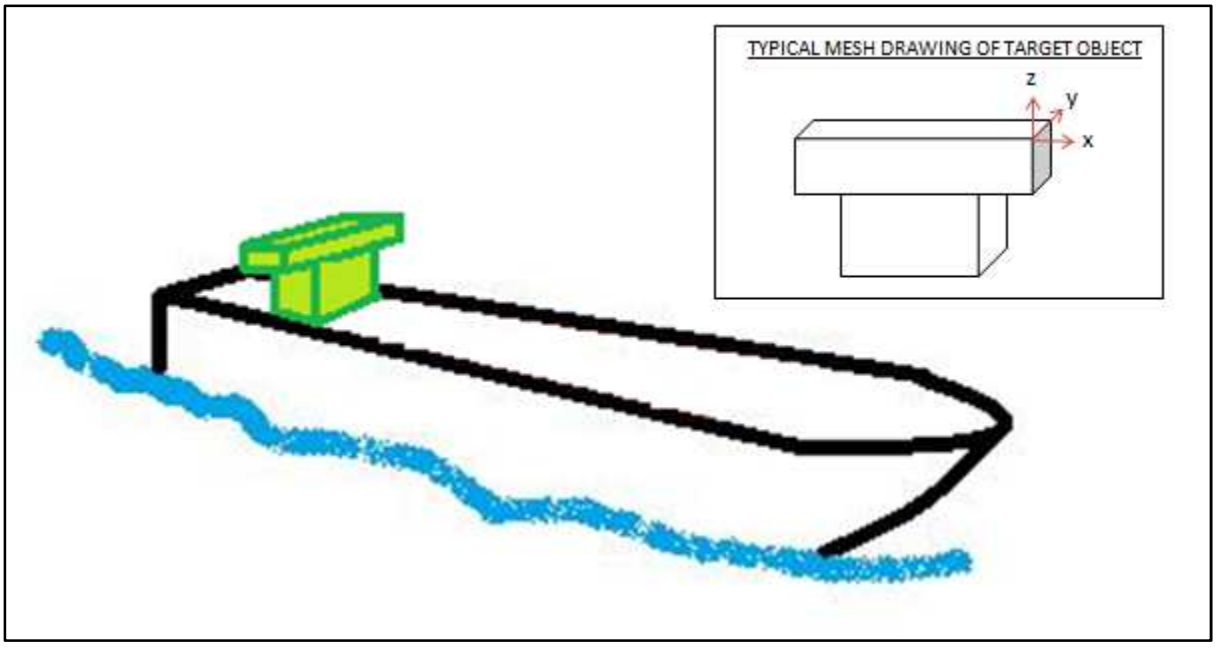

Figure 3.2: Illustration of typical superstructure (in green) being tracked along with its mesh file as input to the tracking algorithm

Table 3.2: Test plan for 3D Object tracking

Test number	Model		Prototype		Test condition
	Hm0 (m)	Tp (s)	Hm0 (m)	Tp (s)	
OBJ01	0.015	1.6	1.5	16	Testing tracking algorithm for “small”, ”smooth” waves
OBJ02	0.015	1.6	1.5	16	Repeatability test for test number 1
OBJ03	0.015	1.6	1.5	16	Repeatability test for test number 1
OBJ04	0.030	1.6	3.0	16	Testing tracking algorithm for “large”, ”smooth” waves
OBJ05	0.030	1.6	3.0	16	Repeatability test for test number 4
OBJ06	0.030	1.6	3.0	16	Repeatability test for test number 4
OBJ07	0.015	1.2	1.5	12	Testing tracking algorithm for “small”, ”jerky” waves
OBJ08	0.015	1.2	1.5	12	Repeatability test for test number 7
OBJ09	0.015	1.2	1.5	12	Repeatability test for test number 7
OBJ10	0.030	1.2	3.0	12	Testing tracking algorithm for “large”, ”jerky” waves
OBJ11	0.030	1.2	3.0	12	Repeatability test for test number 10
OBJ12	0.030	1.2	3.0	12	Repeatability test for test number 10

3.3 Summary

Verification testing will be done for both the 2D LED tracking and 3D Object tracking algorithms, after which system accuracy tests will follow. The results will help better understand the practical usability of both systems for physical model use as well as prototype use in the field.

4 STATIC VERIFICATION OF THE MOTION TRACKING SYSTEM

Comparison of the results during the physical model tests for the 2D LED tracking and 3D object tracking system to that obtained with the Keoship motion capture system, will not cover limitations of the measurement device, i.e. the cameras. For determining the accuracy of the system, a few 'static' tests were conducted. The clearest way to directly measure displacement is by using a Vernier calliper, which can measure distances at 0.04 mm accuracy. The translation results obtained can then be compared with the distances measured using the Vernier calliper.

The 2D LED tracking system and 3D Object tracking system were both tested under controlled conditions to determine the accuracy of the system. The subsections below describe the set-up and results for the verification and accuracy tests.

4.1 Verification Tests

For the verification of the 2D LED tracking software and the 3D Object tracking software the entire deck of the vessel along with the objects to be tracked were statically displaced in the horizontal plane. For these tests, only motion measurements in the horizontal plane (surge, sway and yaw) were possible. The process involved moving or rotating the 2D LED system and 3D cubes, a predetermined distance or angle known to sub-millimeter accuracy, for surge, sway and yaw.

4.1.1 Verification test set-up

For the 2D LED tracking verification tests, the original deck of the model vessel to be used in the physical model tests was curved and therefore not suitable for the placement of a 2D LED rectangle on top of it. The curved deck was therefore replaced by a flat deck. For this, a piece of hardboard, 6 mm thick was measured to size and cut. The piece of hardboard was painted black to minimize any light reflection resulting from any LED's on the deck. A rectangle with size, 500 x 1000 mm was marked out on top of the deck and served as the outline for the LED's. Six individual LED strips, along with wires were glued onto the outline of the rectangle and formed the 2D rectangle to be tracked. An image showing the LED strips glued onto the deck is presented in Figure 4.1.

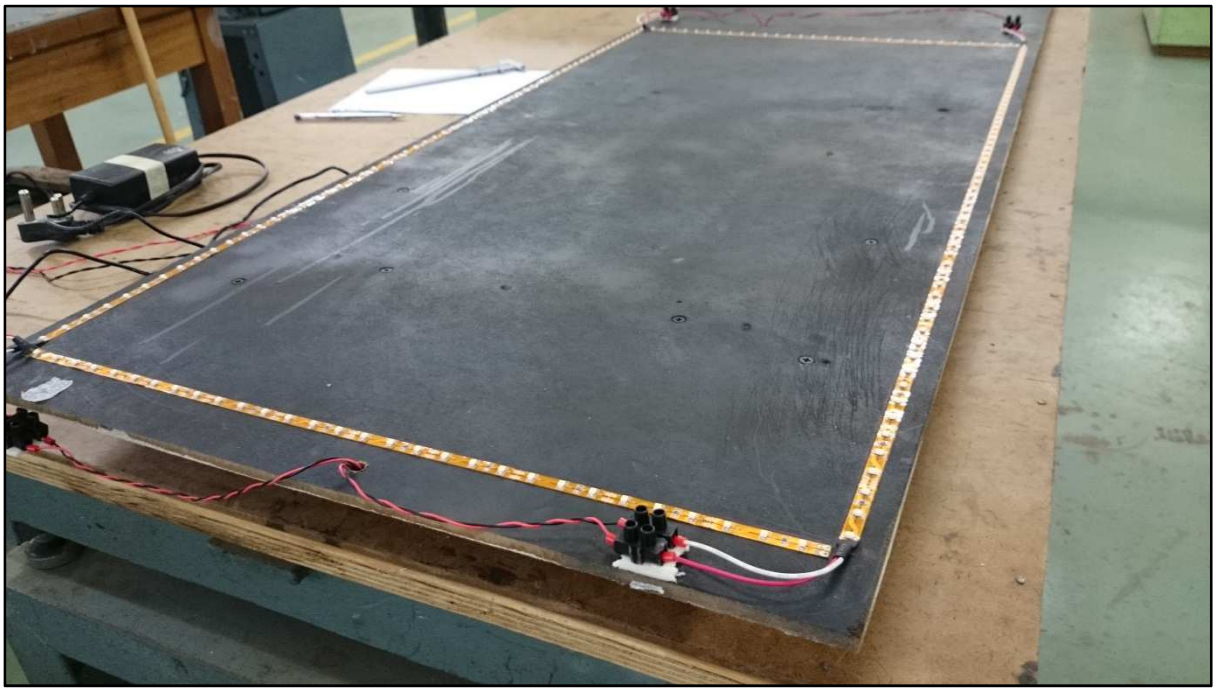

Figure 4.1: LED strips glued in the shape of a rectangle onto the deck

As mentioned in section 3.2.2, for non-intrusive object tracking, the object being tracked will typically be the superstructure of a vessel. For proof-of-concept purposes however, objects with known dimensions were created. Three wooden cubes were built and fixed on top of the vessel deck. The same flat deck which was manufactured for the 2D LED tracking tests was used.

The wooden cubes were all constructed to be 200 x 200 x 200 mm in size. All three cubes were mounted on the longitudinal centre line of the deck. The cubes were mounted on the deck with different orientations. The front and back cubes were fixed with their sides parallel to the longitudinal centre line of the vessel. The cube in the middle was orientated at 45° with respect to the other cubes. Only one cube will be selected and tracked for the verification and physical model tests. The difference in orientation is to ensure that at least one of the three cubes will have little to no light reflection on its sides during a test. Any light reflections might influence the correct working of the tracking algorithm during the development stage. This allows for tracking redundancy during tests.

With the vessel deck being painted black, the cubes were painted white to ensure the best possible tracking for validation purposes. To ensure the tracking algorithm easily picks up the edges of the object to be tracked, they were painted black to stand out. An image showing the object cubes on the deck is presented in Figure 4.2.

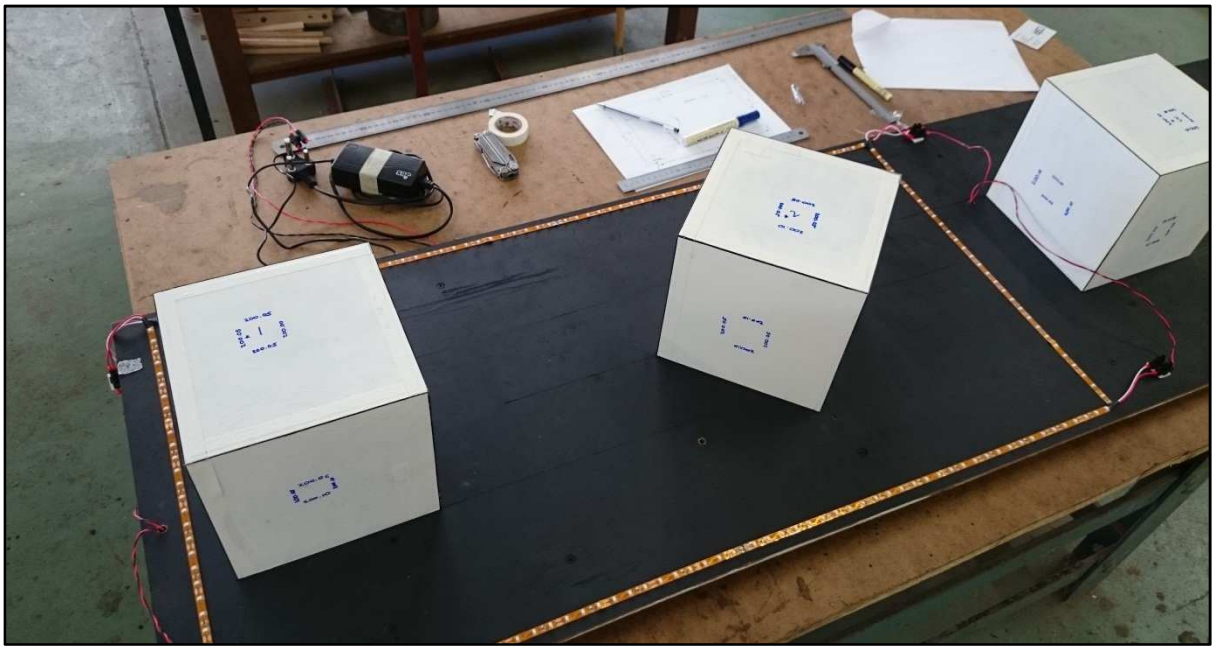

Figure 4.2: Object cubes being placed on the vessel deck

For the set-up, angle irons were clamped onto a plank which formed the horizontal sliding surface. A laser distance meter (make: Leica, model: DISTO A6), along with a Vernier calliper and ruler, were used to set up the angle irons which served as guide rails during the static tests.

Surge and sway movement of a ship were replicated by the sliding of a deck board on the horizontal plane. The deck board representing the deck of the vessel was placed on a horizontal surface, in between some angle irons for the verification tests. Refer to Figure 4.3 for the 2D LED static sliding test set-up. The board was slid in the x-direction for surge and in the y-direction for sway, refer to

Figure 4.4. The width of the gaps between the guide rails and the board were equal to the distance over which the deck board moved when it slid back and forth between the angle irons for both surge and sway.

The sliding distance was accurately determined by measuring the gap, Δx and Δy with a Vernier calliper, refer to Figure 4.5 and Figure 4.6. The same set-up and procedure was used when placing the 3D cube objects onto the deck board.

Yaw movement of a ship was replicated by the rotating of a deck board about a vertical shaft. A vertical rod, 12 mm in diameter was used as the vertical shaft around which rotation would take place. Using a spirit level, the shaft was fixed to be perpendicular to the horizontal plane, refer to Figure 4.7. The deck board was rotated at a known angle about the centre of the object. The rotation results obtained were compared to the rotation angle measured using a compass and the Pythagoras measurement method (Figure 4.8) to verify the accuracy of the measured rotations in the horizontal plane. The horizontal rotation about the shaft for a corner point is illustrated in Figure 4.9.

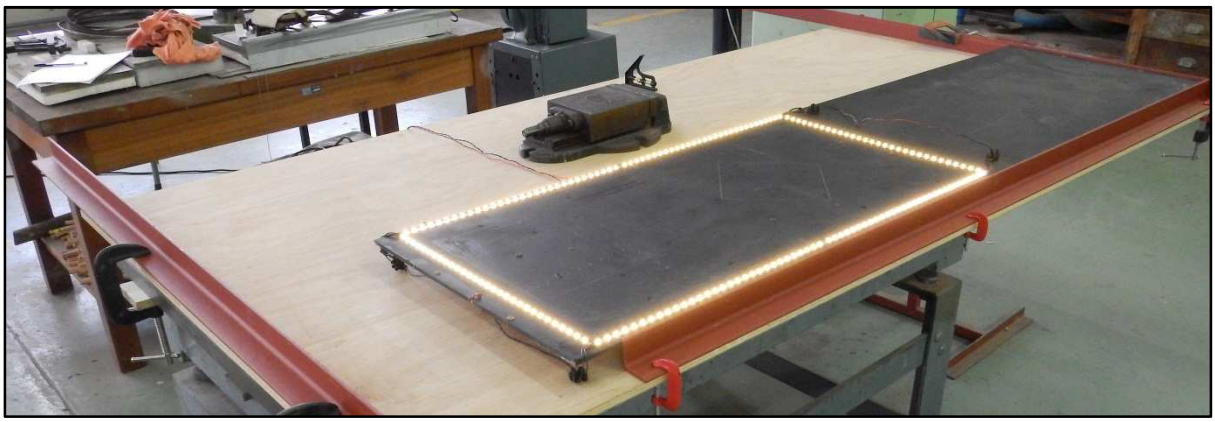

Figure 4.3: Guide rails and 2D LED deck board used for the static surge and sway tests

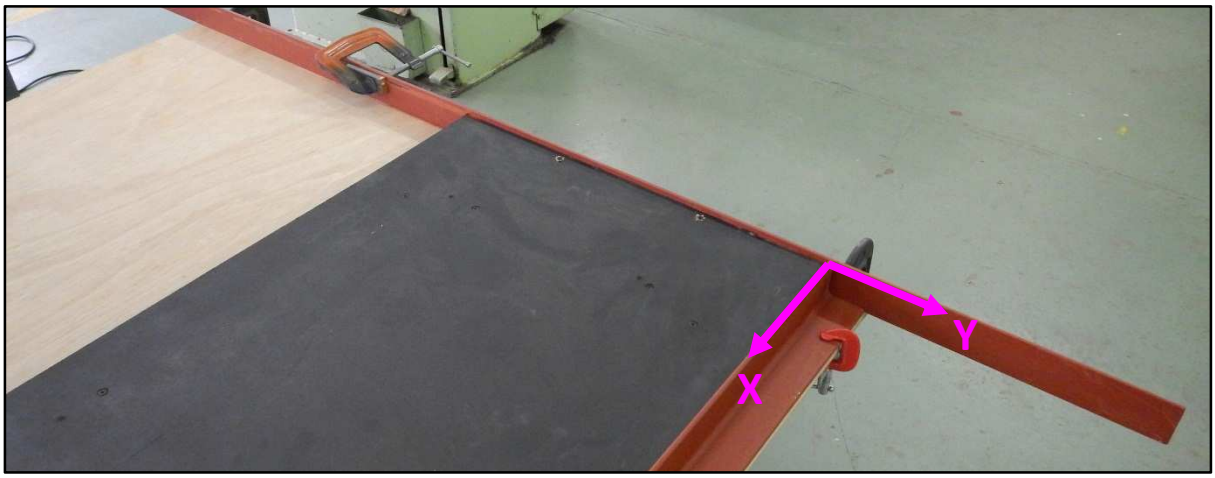

Figure 4.4: Static slide position, with surge and sway both equal to zero

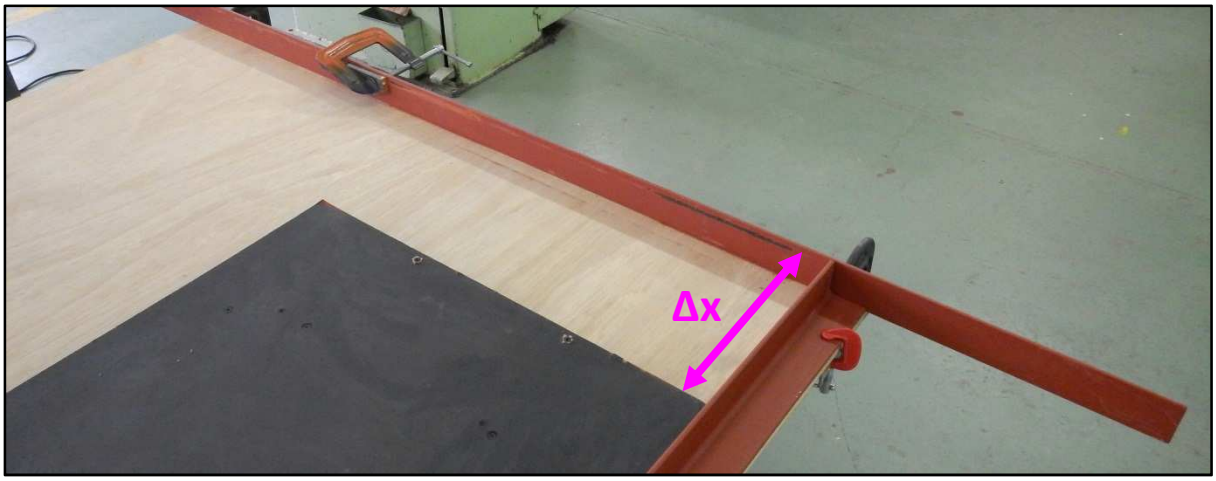

Figure 4.5: Static slide position of the deck board with surge equal to Δx

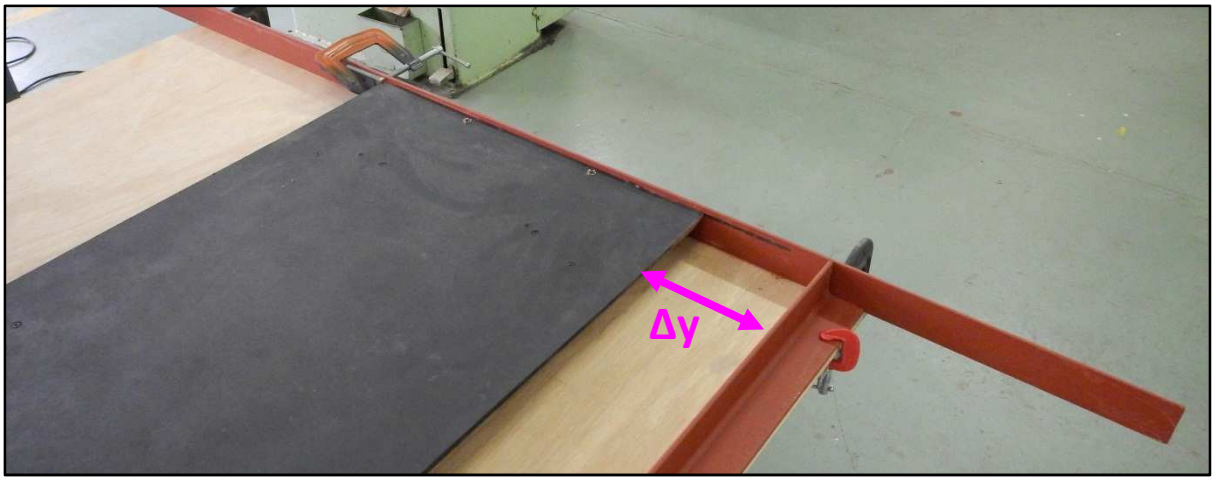

Figure 4.6: Static slide position of the deck board with sway equal to Δy

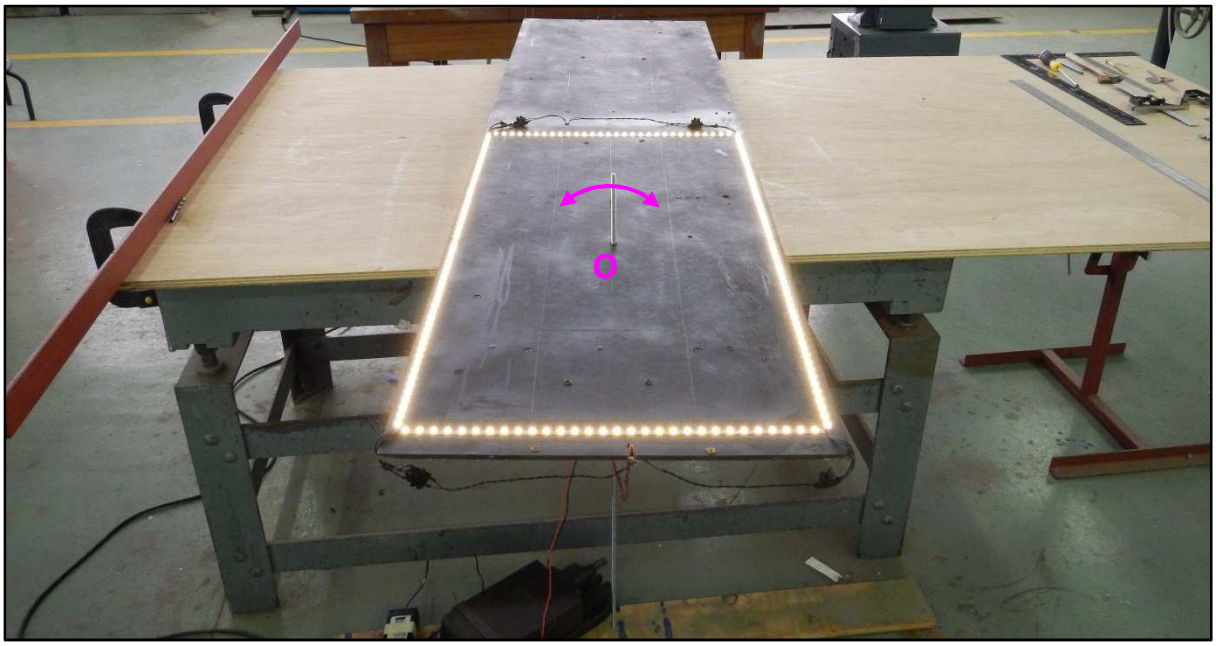

Figure 4.7: Set-up of the 2D LED deck board to rotate about point 'O'

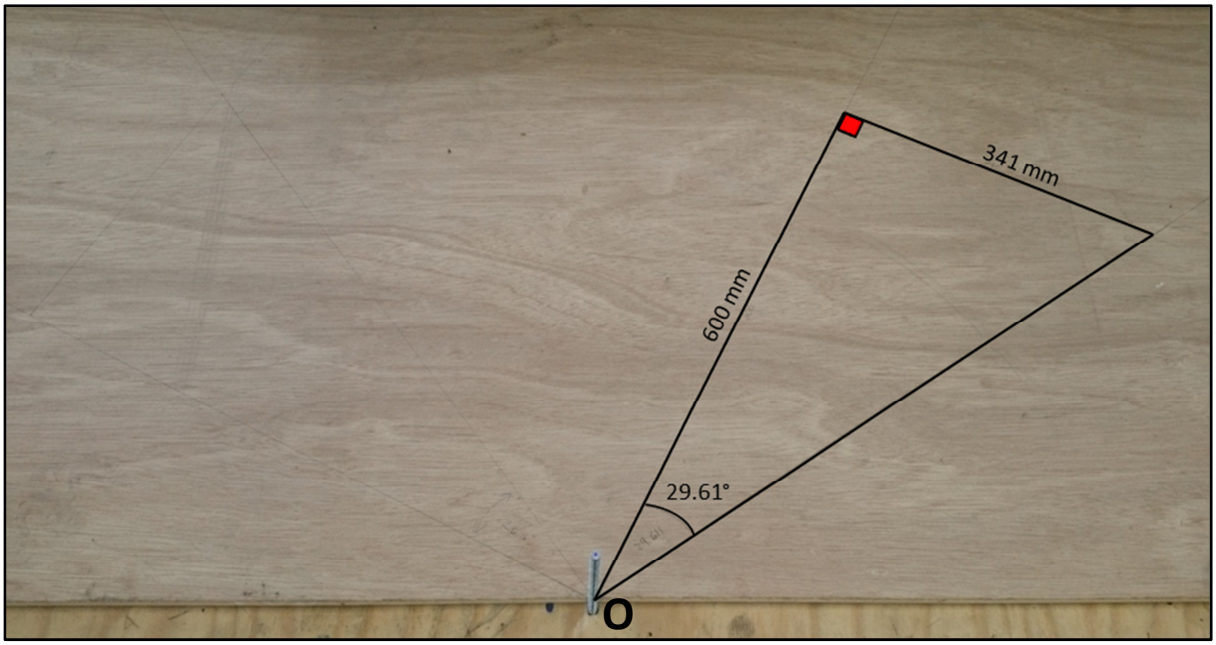

Figure 4.8: Static rotation point 'O' and the rotation angle marked out on a horizontal surface

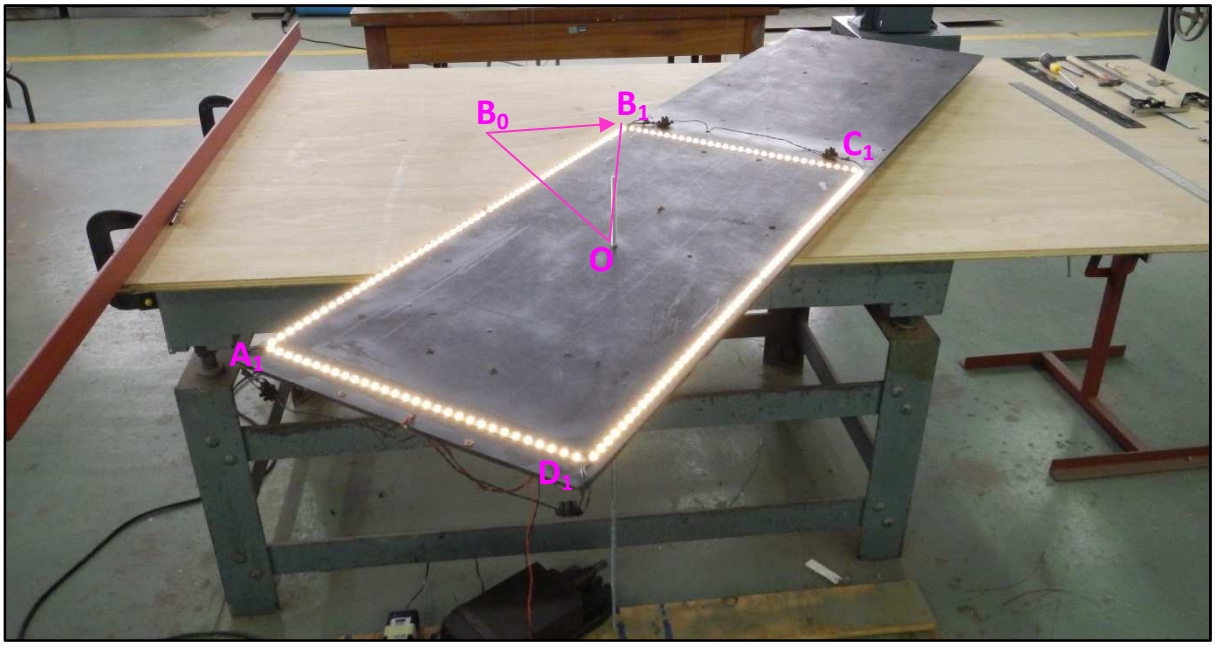

Figure 4.9: illustration of corner point 'B' moving from its original location to a new location when the 2D LED square rotates about its centre Point 'O'

4.1.2 Verification test results

Measurements for surge and sway were done from three different camera view angles (one being perpendicular to the surge direction, two being perpendicular to sway and the third view angle being oblique to both the surge and sway directions).

The measured sliding displacement of the plate is taken as the maximum measured gap between the angle irons for surge and sway. However, the following should be considered:

- There is an error in the direct measurement of around 0.04 mm (i.e. the accuracy of the Vernier calliper).
- From CSIR (2008), which had a similar camera set-up, the standard deviation in the measurement of a still image is roughly 0.04 mm for a similar measurement set-up.

The CSIR (2008) noted that the total possible error is determined as:

$$R_{total} = \sqrt{R_{disp}^2 + R_{vernier}^2 + \sigma_{still\ image}^2} \tag{4.1}$$

Where, R_{disp} is the error in the measured displacement, $R_{vernier}$ is the error in the measurement with the Vernier calliper and $\sigma_{still\ image}$ is the standard deviation in the still image.

The results obtained for the respective view angles are summarised in Table 4.1 and

Table 4.2 for the 2D LED tracking and 3D object tracking respectively.

Table 4.1: Measured sliding displacements for 2D LED tracking static tests

Camera View angle (motion)	Directly measured displacement / angle		Tracking system measurement		R_{disp}
Perpendicular to surge motion (surge)	255.52 mm	(surge)	257.54 mm	(surge)	2.02
	0.00 mm	(heave)	3.23 mm	(heave)	3.23
	0.00 mm	(sway)	4.76 mm	(sway)	4.76
Perpendicular to sway motion (surge)	255.52 mm	(surge)	252.64 mm	(surge)	2.88
	0.00 mm	(heave)	2.70 mm	(heave)	2.70
	0.00 mm	(sway)	0.14 mm	(sway)	0.14
Oblique View angle (surge)	255.52 mm	(surge)	301.61 mm	(surge)	46.09
	0.00 mm	(heave)	39.13 mm	(heave)	39.13
	0.00 mm	(sway)	52.66 mm	(sway)	52.66
Perpendicular to surge motion (sway)	0.00 mm	(surge)	1.58 mm	(surge)	1.58
	0.00 mm	(heave)	0.67 mm	(heave)	0.67
	160.52 mm	(sway)	158.21 mm	(sway)	2.31
Perpendicular to sway motion (sway)	0.00 mm	(surge)	4.65 mm	(surge)	4.65
	0.00 mm	(heave)	2.68 mm	(heave)	2.68
	160.52 mm	(sway)	155.39 mm	(sway)	5.13
Oblique View angle (sway)	0.00 mm	(surge)	25.21 mm	(surge)	25.21
	0.00 mm	(heave)	30.14 mm	(heave)	30.14
	160.52 mm	(sway)	123.27 mm	(sway)	37.25
Oblique View angle (yaw)	0.00°	(roll)	0.05°	(roll)	0.05
	0.00°	(pitch)	0.08°	(pitch)	0.08
	29.61°	(yaw)	29.40°	(yaw)	0.21

Table 4.2: Measured sliding displacements for 3D Object tracking static tests

Motion, (View angle)	Directly measured displacement / angle	Tracking system measurement	R_{disp}
Perpendicular to surge motion (surge)	255.52 mm (surge) 0.00 mm (heave) 0.00 mm (sway)	255.23 mm (surge) 2.95 mm (heave) 0.21 mm (sway)	0.29 2.95 0.21
Perpendicular to sway motion (surge)	255.52 mm (surge) 0.00 mm (heave) 0.00 mm (sway)	256.35 mm (surge) 0.34 mm (heave) 0.87 mm (sway)	0.83 0.34 0.87
Oblique View angle (surge)	255.52 mm (surge) 0.00 mm (heave) 0.00 mm (sway)	258.41mm (surge) 1.73 mm (heave) 2.52 mm (sway)	2.89 1.73 2.52
Perpendicular to surge motion (sway)	0.00 mm (surge) 0.00 mm (heave) 160.52 mm (sway)	1.91 mm (surge) 0.98 mm (heave) 155.31 mm (sway)	1.91 0.98 5.21
Perpendicular to sway motion (sway)	0.00 mm (surge) 0.00 mm (heave) 160.52 mm (sway)	3.28 mm (surge) 0.24 mm (heave) 158.22 mm (sway)	3.28 0.24 2.30
Oblique View angle (sway)	0.00 mm (surge) 0.00 mm (heave) 160.52 mm (sway)	1.14 mm (surge) 0.03 mm (heave) 157.09 mm (sway)	1.14 0.03 3.43
Oblique View angle (yaw)	0.00° (roll) 0.00° (pitch) 29.61° (yaw)	0.12° (roll) 0.12° (pitch) 29.41° (yaw)	0.12 0.12 0.20

The resulting possible error for the 2D LED tracking system over these distances were found to be 52.66 mm for translation and 0.21° for rotation. For the 3D Object tracking system however, this was found to be 5.21 mm for translation and 0.20° for rotation.

Further investigation was done into why the translation errors for both surge and sway were so large for the 2D LED tracking from the oblique view. The investigation revealed a depth estimation error when using the Coplanar POSIT algorithm.

To conform this, the checkerboard was set up a specific distance from the camera, and obliquely angled at 45° to the camera. Four adjacent squares were selected as the first position. The next four adjacent squares were selected as the second position. An illustration of the set-up is shown in Figure 4.10 with the centres of the selected positions marked with red and yellow respectively.

The physical distance between the camera lens and both positions were measured to be 199.5 cm and 203.2 cm respectively. The image coordinates for both positions, along with coplanar object coordinates and calibrated focal length were provided as input to the Coplanar POSIT algorithm. The distance from the camera lens to the centre of each position was estimated by the algorithm as 199.6 cm and 204.9 cm respectively. For the specific set-up the error was 1.6 cm between position locations. The distance between the two positions (before and after), was physically measured as 60.00 mm while the 2D tracking algorithm estimated a displacement equal to 54.82 mm.

The object perspective translation is calculated by converting the camera perspective rotations and translations (which include depth to object) to object translations. The depth estimation error has a minor effect on estimated object translations over small distances. The error becomes quite evident

for the large distances during the verification tests, making the 2D LED tracking algorithm not viable for large motions from an oblique angle.

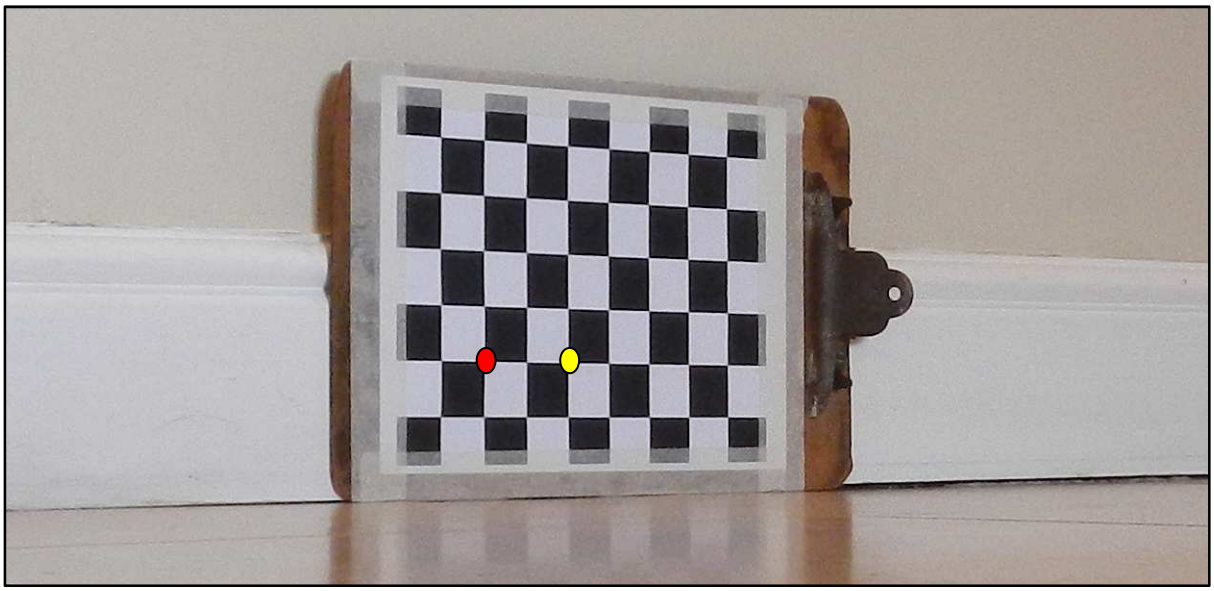

Figure 4.10: Illustration of set-up to determine influence of estimated depth with Coplanar POSIT algorithm

4.2 Accuracy Tests

Determining the accuracy differences over a range of distances for all 6DOF motions would require an intricate set-up when using the same complete hardboard deck as for the verification tests. The results from the verification tests indicate that only the 3D Object tracking system is viable over a range of displacement distances from an oblique angle. For this reason, only the 3D Object tracking system would be used in the accuracy tests over a range of distances, ranging from 10 mm to 1500 mm for translation and 15° to 45° for rotation.

4.2.1 Accuracy test set-up

For the 3D Object tracking accuracy tests, only a single wooden cube was used. To achieve accurate measurements in all 6DOF, the cube being tracked was used completely on its own.

For motions in the horizontal plane, a grid with translation and rotation displacements was created using a computer drawing package and printed onto a 600 mm x 2000 mm sheet of paper. Provision was made for surge and sway movements of 10 mm, 50 mm, 150 mm, 500 mm and 1500 mm as well as yaw movements of 15°, 30° and 45°. Figure 4.11 shows the marked sheet of paper on the horizontal surface previously used for the verification tests.

The set-up for motions in the vertical plane was slightly more intricate. A vertical guide was put in place, onto which the wooden cube was moved up and down to simulate heave motion. The cube was then fixed at pre-marked displacement distances on the vertical guide. Figure 4.12 illustrates the set-up for heave motions. For pitch and roll motions, a mounting bracket usually used for 3D laser scanning purposes was used. The mounting bracket can be adjusted at 15° intervals, accurate to 0.2°. Figure 4.13 shows the wooden cube mounted on the bracket to allow simulated pitch and roll motions.

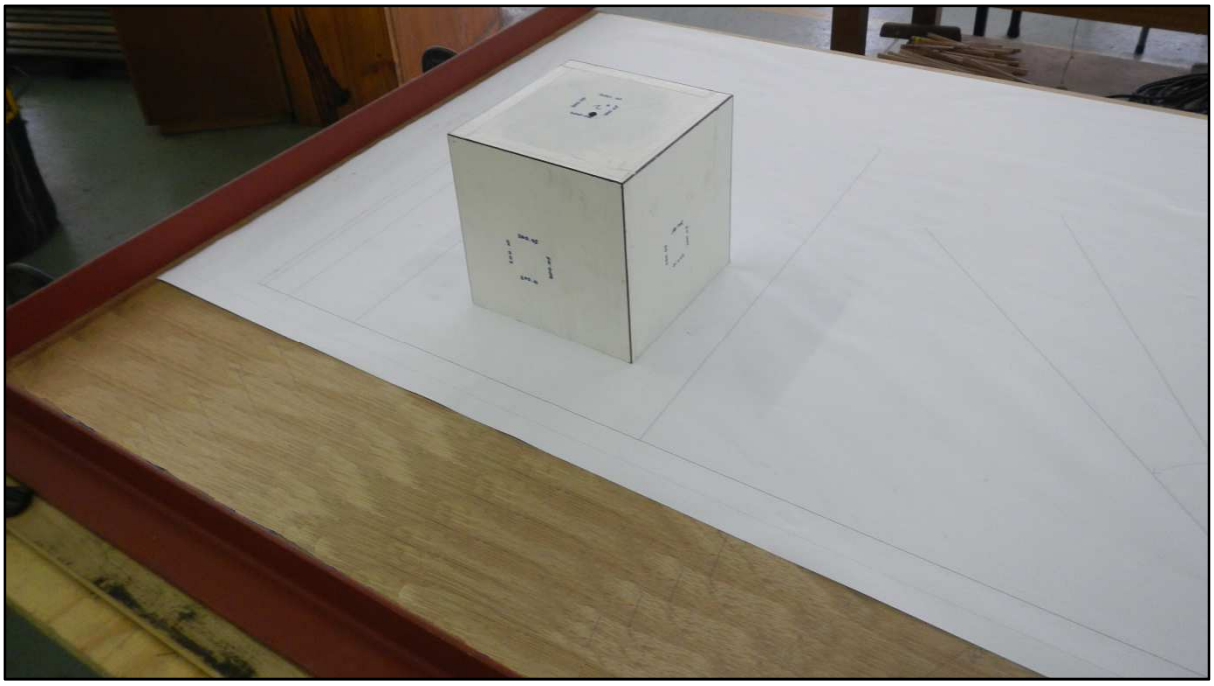

Figure 4.11: Wooden cube on sheet of paper with marked translations and rotations for surge, sway and yaw accuracy tests

Figure 4.12: Vertical guide with wooden cube used for heave accuracy tests

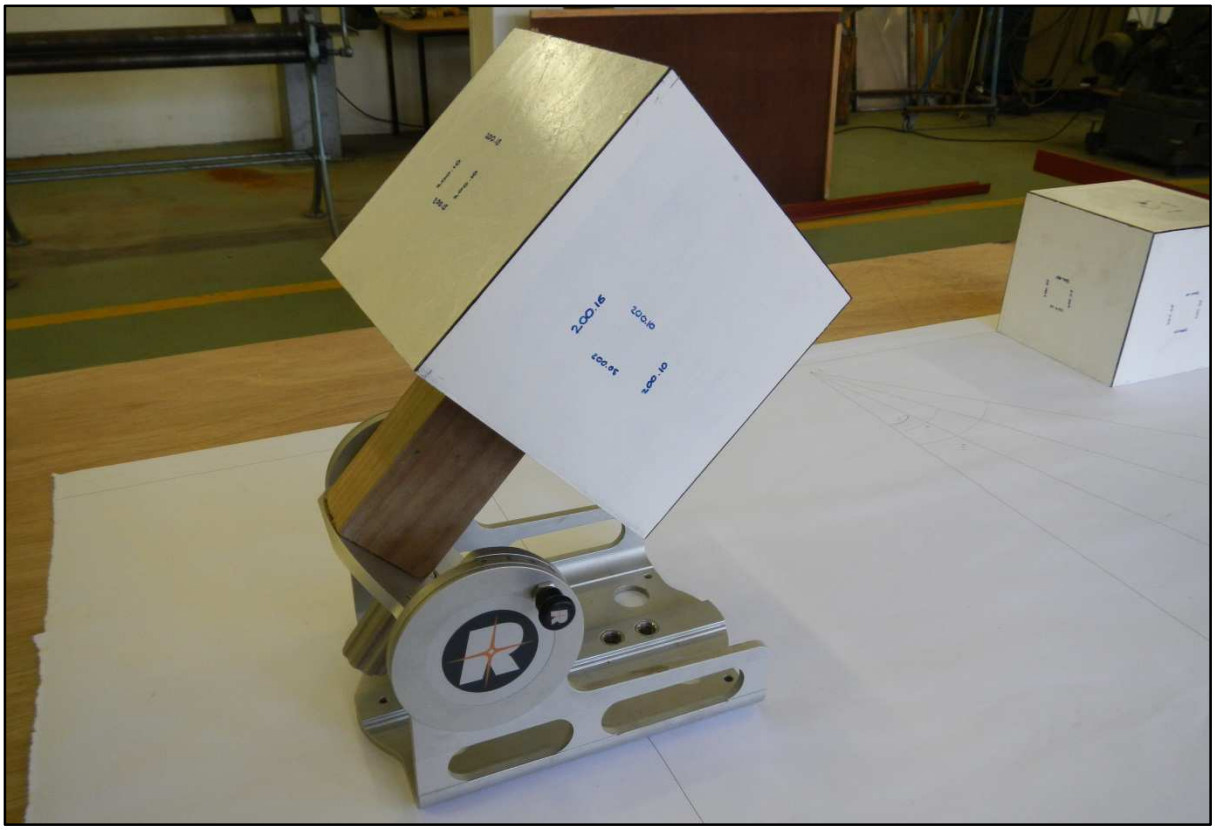

Figure 4.13: Adjustable bracket with wooden cube used for pitch and roll accuracy tests

4.2.2 Accuracy test results

Accuracy measurements for all 6DOF motions were done from an oblique view angle. The percentage error for a range of displacements which ranges from 10 mm to 1500 mm and 15° to 45° were determined. The results obtained for the accuracy tests for the different displacement ranges are summarised in Table 4.3 to Table 4.8 and graphically presented in Figure 4.14 and Figure 4.15.

Table 4.3: Verification of surge motion for 3D object tracking at an oblique angle

Directly measured displacement	Tracking system measurement	R_{disp}	Percentage error (%)
10.00 mm (surge)	10.98 mm (surge)	0.98	9.80
0.00 mm (heave)	0.19 mm (heave)	0.19	-
0.00 mm (sway)	0.03 mm (sway)	0.03	-
50.00 mm (surge)	47.55 mm (surge)	-2.45	4.90
0.00 mm (heave)	0.31 mm (heave)	0.31	-
0.00 mm (sway)	1.94 mm (sway)	1.94	-
150.00 mm (surge)	151.43 mm (surge)	1.43	0.95
0.00 mm (heave)	1.85 mm (heave)	1.85	-
0.00 mm (sway)	0.10 mm (sway)	0.1	-
500.00 mm (surge)	498.60 mm (surge)	-1.4	0.28
0.00 mm (heave)	3.69 mm (heave)	3.69	-
0.00 mm (sway)	3.79 mm (sway)	3.79	-
1500.00 mm (surge)	1507.49 mm (surge)	7.49	0.50
0.00 mm (heave)	10.28 mm (heave)	10.28	-
0.00 mm (sway)	3.17 mm (sway)	3.17	-

Table 4.4: Verification of sway motion for 3D object tracking at an oblique angle

Directly measured displacement	Tracking system measurement	R_{disp}	Percentage error (%)
0.00 mm (surge)	10.98 mm (surge)	2.66	-
0.00 mm (heave)	0.84 mm (heave)	0.84	-
10.00 mm (sway)	2.66 mm (sway)	0.98	9.80
0.00 mm (surge)	2.36 mm (surge)	2.36	-
0.00 mm (heave)	0.39 mm (heave)	0.39	-
50.00 mm (sway)	51.88 mm (sway)	1.88	3.76
0.00 mm (surge)	4.27 mm (surge)	4.27	-
0.00 mm (heave)	0.45 mm (heave)	0.45	-
150.00 mm (sway)	153.85 mm (sway)	3.85	2.57
0.00 mm (surge)	1.36 mm (surge)	1.36	-
0.00 mm (heave)	4.47 mm (heave)	4.47	-
500.00 mm (sway)	502.41 mm (sway)	2.41	0.48
0.00 mm (surge)	3.25 mm (surge)	3.25	-
0.00 mm (heave)	13.25 mm (heave)	13.25	-
1500.00 mm (sway)	1507.49 mm (sway)	7.49	0.50

Table 4.5: Verification of heave motion for 3D object tracking at an oblique angle

Directly measured displacement	Tracking system measurement	R_{disp}	Percentage error (%)
0.00 mm (surge)	3.33 mm (surge)	3.33	-
10.00 mm (heave)	15.01 mm (heave)	5.01	50.10
0.00 mm (sway)	7.68 mm (sway)	7.68	-
0.00 mm (surge)	1.28 mm (surge)	1.28	-
50.00 mm (heave)	54.13 mm (heave)	4.13	8.26
0.00 mm (sway)	6.01 mm (sway)	6.01	-
0.00 mm (surge)	6.98 mm (surge)	6.98	-
150.00 mm (heave)	148.45 mm (heave)	-1.55	1.03
0.00 mm (sway)	1.44 mm (sway)	1.44	-
0.00 mm (surge)	19.95 mm (surge)	19.95	-
700.00 mm (heave)	699.71 mm (heave)	-0.29	0.04
0.00 mm (sway)	2.79 mm (sway)	2.79	-
0.00 mm (surge)	18.06 mm (surge)	18.06	-
1500.00 mm (heave)	1503.19 mm (heave)	3.19	0.21
0.00 mm (sway)	7.31 mm (sway)	7.31	-

Table 4.6: Verification of roll motion for 3D object tracking at an oblique angle

Directly measured angle	Tracking system measurement	R_{disp}	Percentage error (%)
15.00° (roll)	14.99° (roll)	-0.01	0.07
0.00° (pitch)	0.79° (pitch)	0.79	-
0.00° (yaw)	0.63° (yaw)	0.63	-
30.00° (roll)	29.69° (roll)	-0.31	1.03
0.00° (pitch)	1.31° (pitch)	1.31	-
0.00° (yaw)	1.77° (yaw)	1.77	-
45.00° (roll)	44.59° (roll)	-0.41	0.91
0.00° (pitch)	1.50° (pitch)	1.50	-
0.00° (yaw)	2.62° (yaw)	2.62	-

Table 4.7: Verification of pitch motion for 3D object tracking at an oblique angle

Directly measured angle	Tracking system measurement	R_{disp}	Percentage error (%)
0.00° (roll)	0.88° (roll)	0.88	-
15.00° (pitch)	15.45° (pitch)	0.45	3
0.00° (yaw)	0.70° (yaw)	0.7	-
0.00° (roll)	1.68° (roll)	1.68	-
30.00° (pitch)	30.37° (pitch)	0.37	1.23
0.00° (yaw)	1.03° (yaw)	1.03	-
0.00° (roll)	2.71° (roll)	2.71	-
45.00° (pitch)	45.08° (pitch)	0.08	0.18
0.00° (yaw)	1.06° (yaw)	1.06	-

Table 4.8: Verification of yaw motion for 3D object tracking at an oblique angle

Directly measured angle	Tracking system measurement	R_{disp}	Percentage error (%)
0.00° (roll) 0.00° (pitch) 15.00° (yaw)	0.14° (roll) 0.24° (pitch) 15.06° (yaw)	0.14 0.24 0.06	- - 0.40
0.00° (roll) 0.00° (pitch) 30.00° (yaw)	0.21° (roll) 0.29° (pitch) 30.18° (yaw)	0.21 0.29 0.18	- - 0.60
0.00° (roll) 0.00° (pitch) 45.00° (yaw)	0.29° (roll) 0.15° (pitch) 45.17° (yaw)	0.29 0.15 0.17	- - 0.38

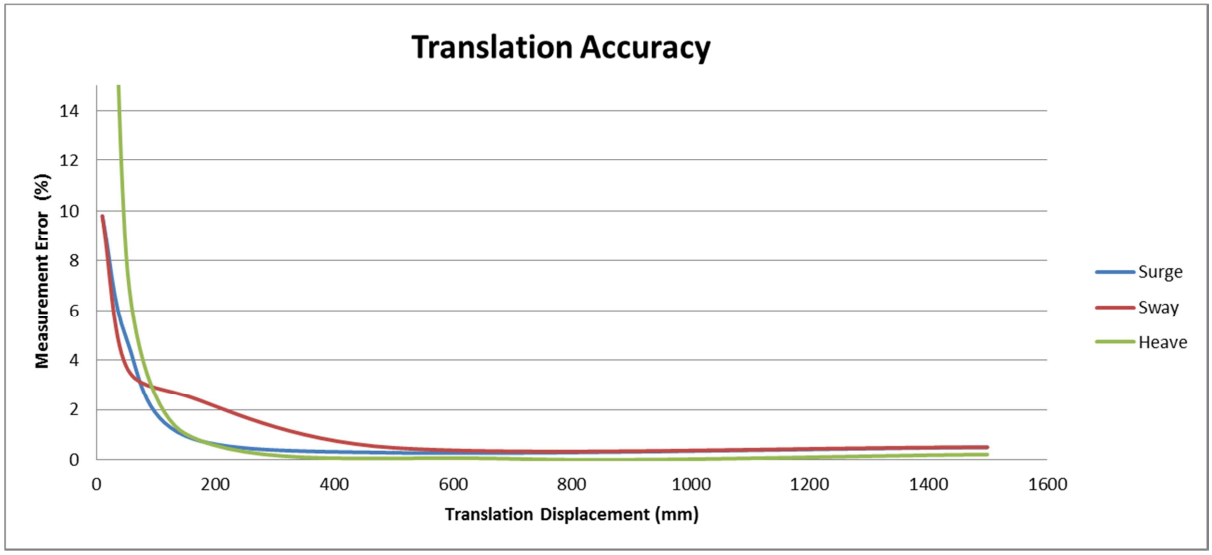

Figure 4.14: Graphical representation of translation accuracy test results

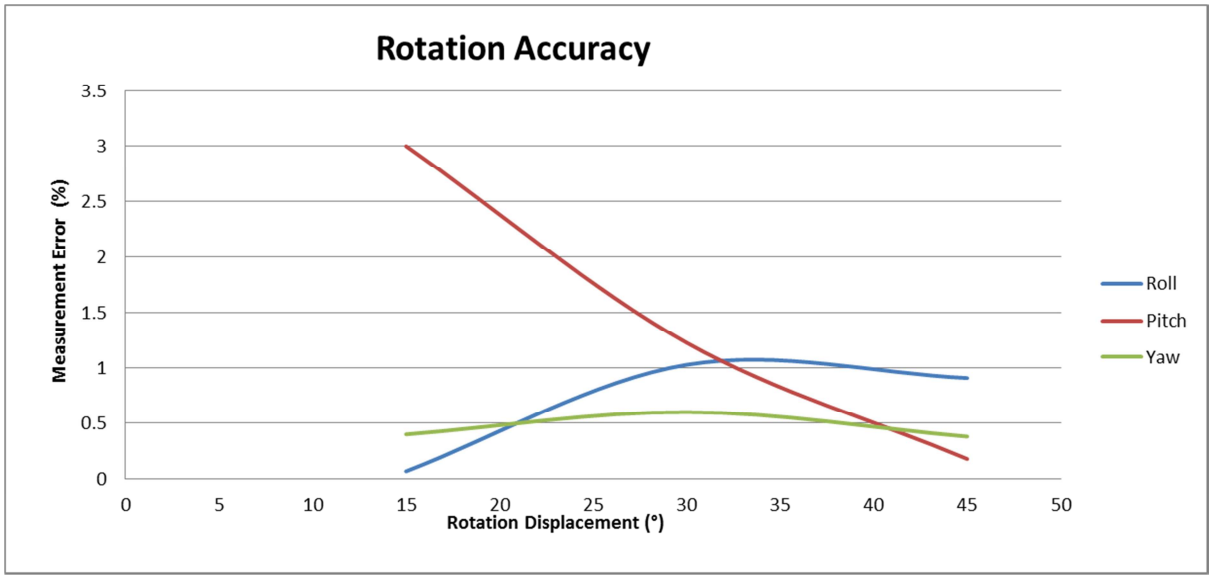

Figure 4.15: Graphical representation of rotation accuracy test results

The accuracy results indicate that the 3D tracking system accurately estimates surge and sway for translation.

Measurement of heave motion in the vertical plane, is estimated the least accurate of the 6DOF motions. Rotation and pitch motion, which are also in the vertical plane are measured less accurately than roll and yaw. The results also indicate that roll and yaw motions are more accurately measured for small rotations.

4.3 Summary

The translation error resulting from the 2D LED tracking algorithm becomes quite evident for the large distances during the verification tests (i.e. displacement distances equal to 160.52 mm and 255.52 mm), making it not viable for large motions.

The results from the accuracy tests done on the 3D Object tracking system showed a drastic decrease in the percentage estimation error when motions are incrementally enlarged. Estimating surge motion for example, improved in accuracy from 9.8% to 0.5% for a displacement of 10 mm and 1500 mm respectively.

Although it was found that the 2D LED tracking system is not viable for large motion measurements, it will still be validated along with the 3D Object tracking system when conducting the physical model tests.

5 PHYSICAL MODEL CONSTRUCTION AND TEST VESSEL

The model infrastructure and bathymetry used were already in place as it was the remainder of a previous physical model project which was conducted at the hydraulics laboratory at the CSIR. The model system contains one model ship and one piled dock. Figure 5.1 and Figure 5.2 represent drawings of the plan and side views of the piled dock in prototype. In the model the piled dock is made up of a flat sheet of steel, specifically sized and secured in place with treaded rods.

To ensure that the motion results gathered will be a true reflection of a typical vessel motion response study, it was decided to test with similar wave conditions and model ship to that of the vessel motion project for which the bathymetry was originally constructed.

The model scale along with model infrastructure and construction particulars are provided in the subsections below.

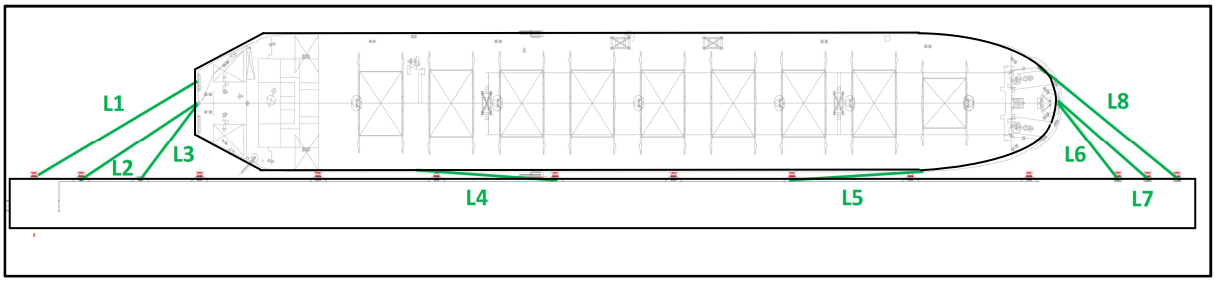

Figure 5.1: Plan view of piled dock to be modelled

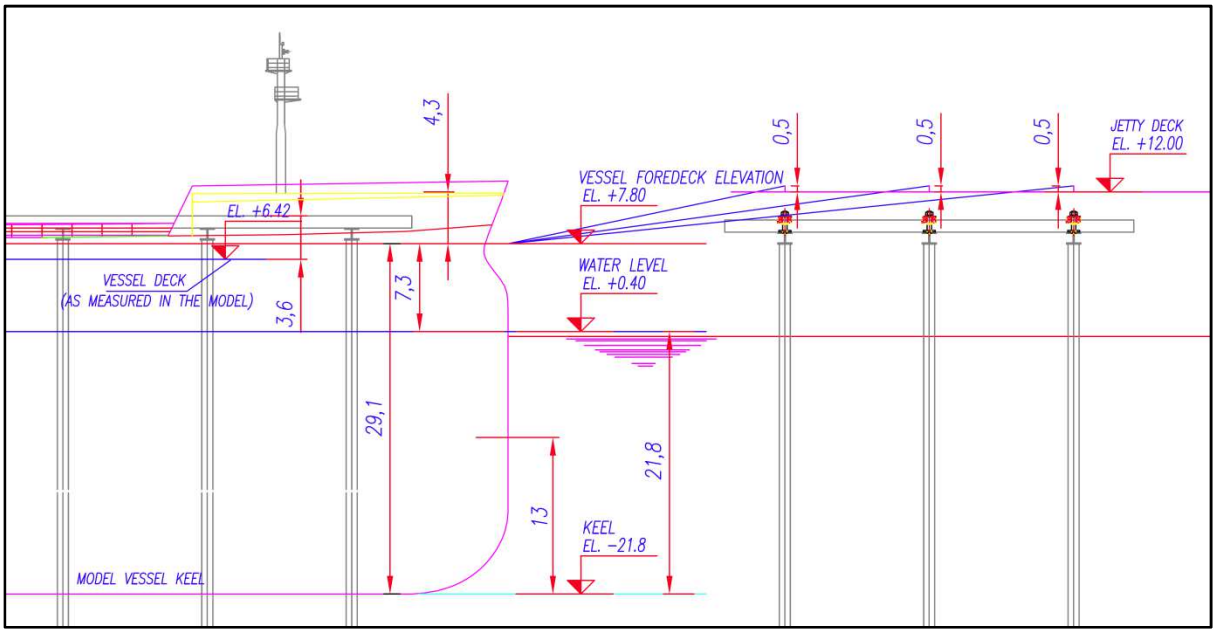

Figure 5.2: Side view of mooring connections and piled dock to be modelled

5.1 Scaling

When scaling a physical model, all physical processes should ideally be correctly scaled. This is however not always possible due to practical reasons. In most cases, it is also not necessary to scale all processes involved; only the dominating processes of interest are required.

In the physical model all lengths are scaled by a certain factor. Scaled models are typically constructed to be as large as possible to minimise scale effects. Not all physical properties can be scaled down at the same scale, causing unwanted scaling effects. In hydraulic models, most scale effects occur due to the properties of water (i.e. density, viscosity and surface tension) which are not scalable (Hughes 1993).

For this model it is assumed that energy dissipation, due to friction with the basin floor, is not significant for the small distance the waves travel in the model. Similarly, skin friction between the water and the hull is ignored. Therefore the viscosity effects are neglected in the scaling. Gravity and inertia are the dominating forces driving the waves and the motion of the ship. In setting up the model, it was therefore decided to ensure similitude with the Froude law of scaling. The Froude law conserves both gravity and inertial forces both of which, being relevant to the previously constructed ship motions model, now form the foundation for the current study.

The model was constructed at a geometric scale of 1:100 (model:prototype). The scaling laws result in the scaling factors provided in Table 5.1, where L_{prot} and L_{mod} are the length units in prototype and model respectively. While ρ_{prot} and ρ_{mod} relates to prototype and model water densities.

Table 5.1: Scaling factors (prototype/model) (Hughes 1993)

Variable	Scale factor
Length of distance [m]	$N_L = \frac{L_{prot}}{L_{mod}} = \frac{100}{1} = 100$
Time [s]	$N_T = \sqrt{\frac{L_{prot}}{L_{mod}}} = \sqrt{\frac{100}{1}} = 10$
Volume [m^3]	$N_V = \left(\frac{L_{prot}}{L_{mod}}\right)^3 = \frac{100^3}{1} = 1000000$
Mass Density [kg/m^3]	$N_\rho = \frac{\rho_{prot}}{\rho_{mod}} = \frac{1025}{1000} = 1.025$
Specific Weight [kg/m^2s^2]	$N_\gamma = N_\rho N_g = \frac{1025}{1000} = 1.025$ (for all practical purposes, the gravitational scale, N_g is unity)
Mass [m^3]	$N_M = {N_L}^3 N_\rho = 1025000$
Force [N]	$N_F = {N_L}^3 N_\gamma = 1025000$

5.2 Wave Basin and Bathymetry

The seabed bathymetry and piled dock was constructed at a scale of 1:100. Previous physical model tests for moored ship motion studies have shown that scales ranging from 1:80 to 1:120 yield acceptable results (Ligteringen et al 2001, CSIR et al. 2008). Figure 5.3 shows the wave basin with constructed model.

Figure 5.3: Wave basin with constructed model

Seaward of the modelled bathymetry is a 1:20 transition slope. This was created to link the flat bed in front of the wavemaker to the bathymetry. The rather gentle slope was chosen so that the long waves can grow gradually as the short waves travel over the transition slope. Moreover, unrealistic 3D effects are minimised with a gentle slope. The distance of 4.5 m (model) between the top of the transition slope and the edge of the departure basin was considered sufficient to form a realistic wave-field.

5.3 Wave Generator, Wave guides and Absorption beaches

In the experiment the waves were generated by a 24 m bank of multi-element wavemakers. The HR Wallingford piston-type wave generation system is driven by a rack-and-pinion mechanism and consists of 48 individual paddles with a width of 0.50 m. The wave maker is equipped with active wave absorption. This was however turned off in the model tests, because the required absorption of very small long waves at the wavemaker would create more noise than a reduction in reflected wave energy.

Solid wave guides were placed at the sides of the wavemakers running parallel up the transition slope stretching 8 m into the model bathymetry. These wave guides are taken into account by the wave generation software so that a homogeneous wave-field is achieved. The area, past the wave guides is sufficient to allow for waves to naturally refract from the seabed profile and ultimately break at the absorption beach at the back of the model. Figure 5.4 shows the completed 24 m bank of wavemakers and a portion of the solid waveguides.

Figure 5.4: Complete 24 m wavemaker bank with wave guides

5.4 Model Vessel & Piled dock

The moored ship motion measurements have been simulated on a fully laden 300 kdwt ship. The model vessel represents a typical large 300 kdwt bulk carrier that calls on South Africa's bulk ore ports. The characteristics of the loaded model vessel are given in Table 5.2.

Length dimensions are given in Figure 5.5 and Figure 5.6. The model vessel was calibrated at even keel, using a cradle as shown in Figure 5.7. During the calibration process, the model vessel was placed in a cradle, which only allows pitching motions. This is done to determine the longitudinal moment of inertia and the height of the centre of gravity. The vessel, plus cradle system, can swing around the cradle axis with little friction. The period of free oscillation was measured. Thereafter the longitudinal moment of inertia for the vessel was calculated given the known mass of the cradle, mass of the vessel and the moment of inertia for the cradle. The achieved longitudinal moment of inertia was equal to that used for the same vessel in previous model studies at the CSIR. This was achieved by shifting ballast blocks in the hull until correct.

The roll period was measured by placing the model vessel in an open water basin at a water depth equal to that of the average water depth in the berth pocket. The process involved measuring a number of free oscillations in the roll direction. The ballast blocks were then shifted laterally until the transverse moment of inertia achieved was equal to that used for the same vessel in previous model studies at the CSIR. Both the longitudinal and transverse moment of inertia is usually given as loading parameter by a client. The calibration process incorporates general vessel stability equations.

Table 5.2: Model vessel parameters

Description	Symbol	B300 prototype	B300 scale model
Length over all	L_{oa}	337 m	3.37 m
Length between perpendiculars	L_{pp}	326 m	3.26 m
Beam	B	54 m	0.54 m
Draught	T	21.0 m	0.21 m
Displacement volume	∇	290109 m^3	0.290109 m^3
Displaced weight	Δ	299 983 t	299.983 kg
Cargo tonnage	DWT	299 983 t	299.983 kg
Height of centre of gravity (CG), (m) above Keel	KG	13.2 m	0.132 m
Distance of CG forward of midships	LCG	10 m	0.1 m
Transverse radius of gyration	k_{xx}	19.8 m	0.198 m
Longitudinal radius of gyration	k_{yy}	89.2 m	0.892 m

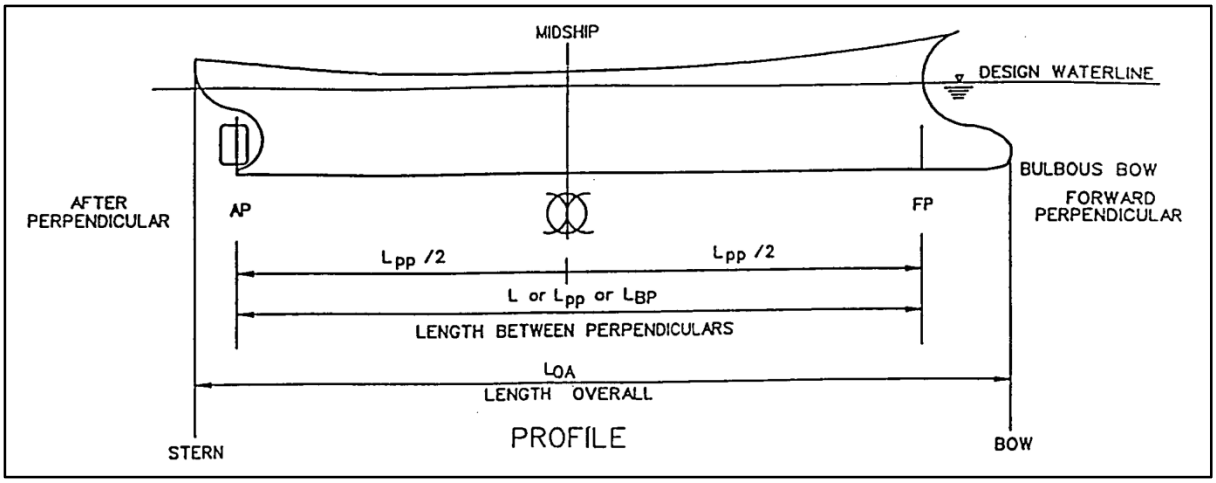

Figure 5.5: Longitudinal cross section of ship with dimension definitions (USACE 2006)

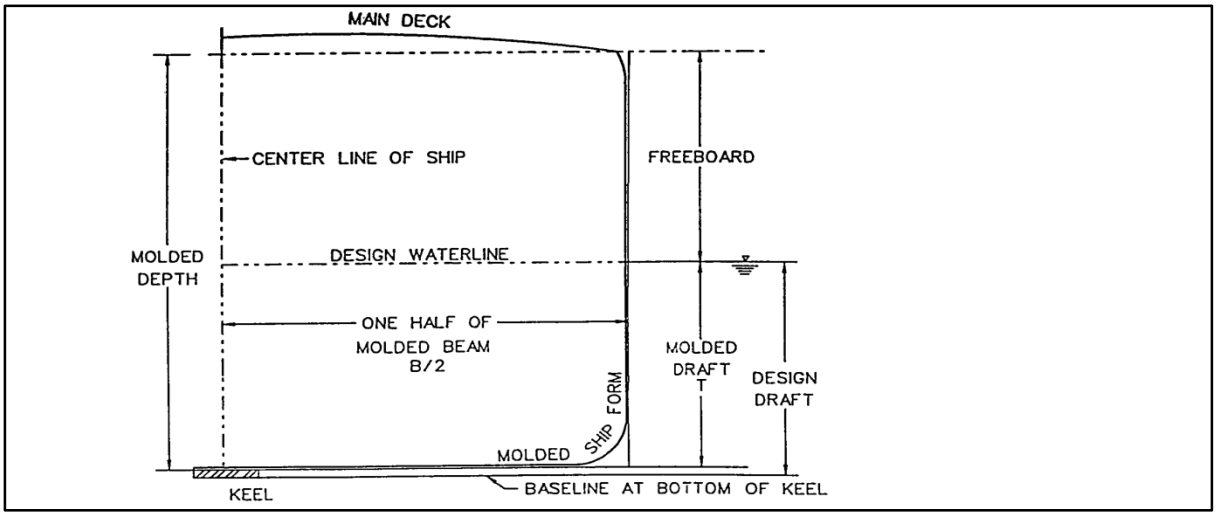

Figure 5.6: Transverse cross section of ship with dimension definitions (USACE 2006)

Figure 5.7: Model vessel in calibration cradle

A piled dock was constructed only after the construction of the bathymetry was completed. The location of the piled dock was surveyed on top of the finished bathymetry. Holes were drilled in the required positions and threaded rods were cemented vertically into place. A 5 mm thick steel plate, which modelled the deck of the piled dock, was secured on the threaded rods and surveyed to the correct height.

6 PHYSICAL MODEL INSTRUMENTATION AND CALIBRATION

The instrumentation used for the generation of waves as well as the different data acquisition (DAQ) devices and software used to log measurements during a test, are described in the subsections below.

6.1 Wave Generation and Calibration

For simplicity, the wave conditions tested were similar to the wave conditions used during the last vessel motion project for which the bathymetry was originally constructed. For the current testing, only the amplitudes were adjusted to ensure that the amplitude margin between the two different wave heights tested (small and large) was sufficient.

A single wave probe, located 1.5 m in front of the bow, was used as the target location during the calibration process. During calibration of the wave generation, the model ship was removed from the model to ensure that the wave measurements would not be corrupted by waves reflecting from the ship. Each of the simulated wave conditions were achieved in the physical model by modifying the gain factor which controls the wave amplitude. Control software which comes standard with the wavemakers from HR Wallingford was used to generate the waves.

6.2 Wave Measurement and Acquisition System

Wave gauges transfer the water surface elevation to electrical measurements. For this model, capacitance gauges were used. These gauges measure the capacitance between a thin insulated wire and the surrounding water which acts as the ground. The measured capacitance is proportional to the height of the water column between the wires.

All the probes were calibrated daily before testing. The calibration process is done to relate water surface fluctuations during tests to wave parameters. The process entailed varying water levels at the probes under controlled conditions and measuring voltages corresponding to each specific water level. A calibration constant was then derived for the entire length of each probe.

The output datasets captured from the capacitance probes were analysed using GEDAP analysis software from CHC (Miles, 1997). During the analysis, the voltage data is converted to a time-series of the variation in the wave surface elevation, from which the wave parameters are calculated.

6.3 Keoship Motion Capture System

The Keoship method was set up as the reference in the validation method, to which the developed motion measurement methods will be compared.

Standard CSIR procedures were followed in setting up the Keoship system. Figure 6.1 shows a Keogram plate on the bow with painted black and white sectors used for tracking purposes while Figure 6.2 shows a view of the camera focusing on the stern. A side view of the ship with the bow and stern deck viewed in the mirrors which were placed on the jetty is given in Figure 6.3.

The computer that was connected to the cameras was set-up to track the black/white interfaces on the Keogram plates. The interfaces at the port and starboard sides of the ship are used to monitor the vertical ship motions, i.e. heave, roll and pitch while, the interfaces on the deck, viewed in the mirror, are used to monitor the horizontal ship motions, i.e. surge, sway and yaw.

Figure 6.1: 300 kwdt model vessel with Keogram plate at the bow

Figure 6.2: View of the Keogram camera focusing on the stern of the model vessel with the Keogram sampling lines (view of deck plate is shown in mirror)

Figure 6.3: 300 kwdt model vessel with mirrors at the jetty to view on the deck

6.4 Force Measurement Acquisition System

Besides setting up the Keoship system as the validation method, to which the developed motion measurement methods will be compared, the forces in mooring lines and fenders will be measured as well.

The intention of the force measurements is only to serve as backup for the Keoship system when abnormalities are thought to be present in the Keoship measurements. The force measurement system of the CSIR, which was installed on the jetty, does however not measure forces on all the fenders installed.

Thus, direct comparison between the calculated ship motions from the strain gauge system to that of ship motions directly obtained from the keoship system will not be possible. Rather, the mooring line and fender forces calculated from the measured ship motions can be compared with the directly measured forces which are available by using strain gauges at the model fenders and bollard blocks.

Standard CSIR procedures which the author helped to developed in the last few years, were followed in setting up and calibrating the force measurement system. A plan layout of the fender and bollard locations is given in Figure 6.4. A picture of a mooring line with bollard block and pulley system is given in Figure 6.5.

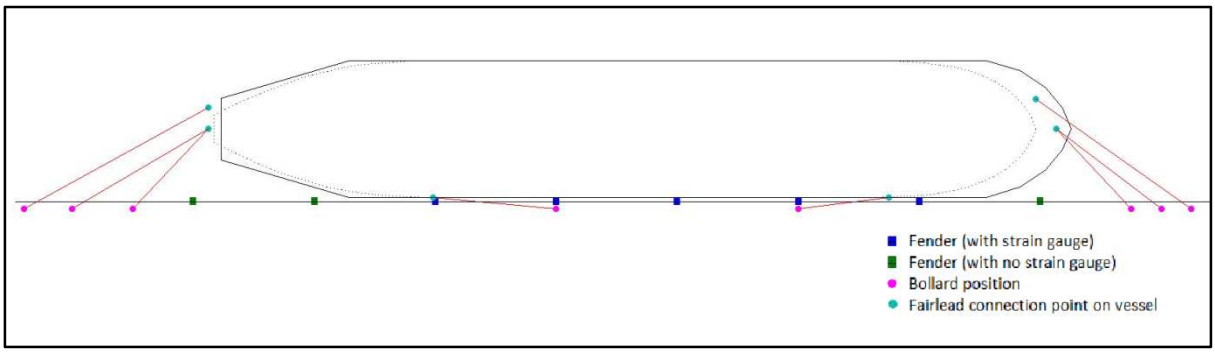

Figure 6.4: Plan layout showing bollard and fender locations

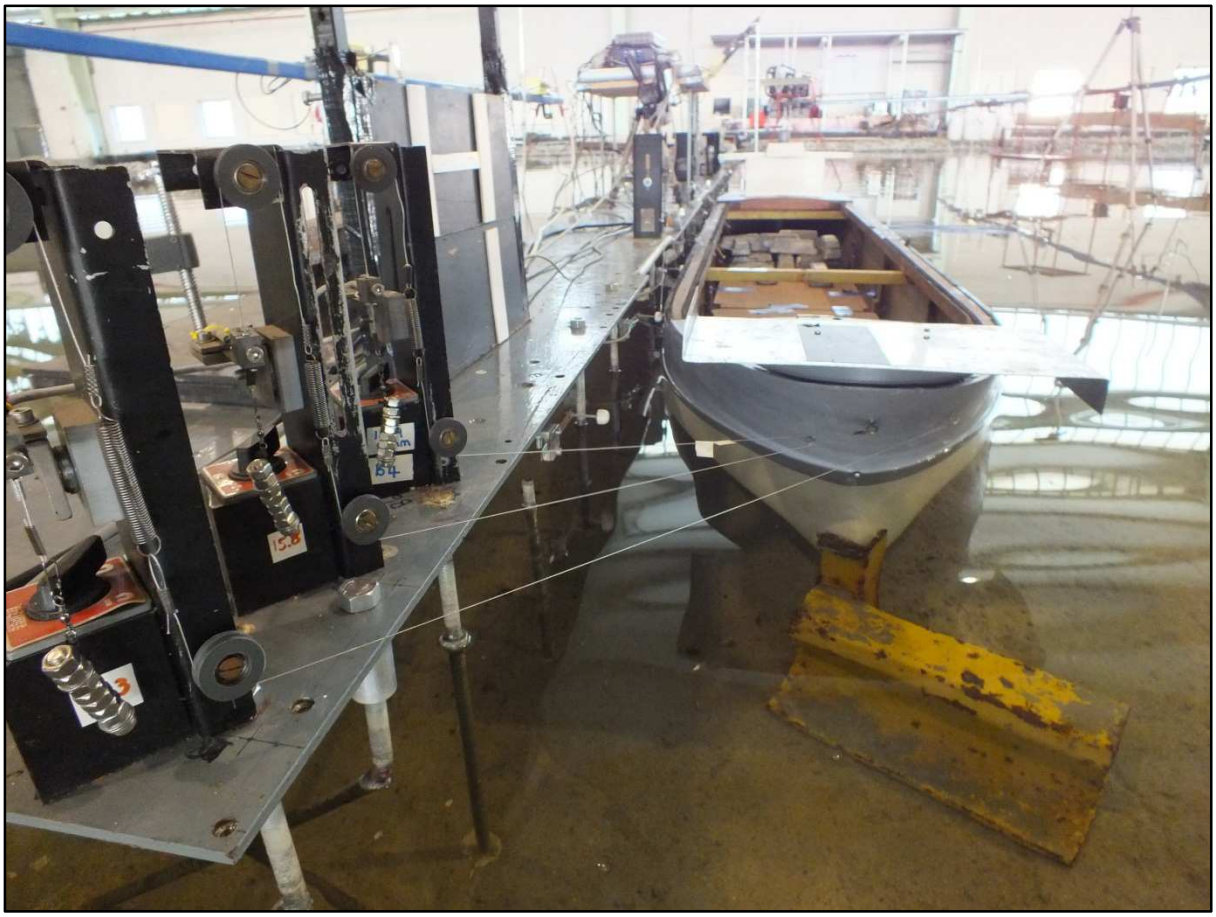

Figure 6.5: Mooring line with bollard block and pulley system for the model set-up

6.5 2D LED Square and 3D Object Tracking System

The new experimental video monitoring tracking system, which is validated in this dissertation, was developed by the author. Although the proof-of-concept system only uses a single camera, a second camera was also set up during the physical modelling testing. The second camera was set up with a different view angle than the first and allowed for system redundancy if one failed during testing.

This section describes the hardware overview and set-up of both the 2D and 3D systems on the vessel.

6.5.1 Hardware overview

The system is composed of two independently set-up commercial cameras with a High Definition (HD) sensor resolution. The cameras used have a 1920×1080 pixels resolution Complementary Metal-Oxide Semiconductor (CMOS) sensor. The cameras were set to record full HD video files at a 1920x1080 pixel resolution and 30 frames per second (fps). The manufacturer technical specification when set to record full HD video files is shown in Table 6.1.

Table 6.1: Technical specifications of the consumer-grade cameras when set to record full HD video files

Camera Model	Nikon AW110 and AW100
Sensor type	1/2.3 inch, CMOS Sensor
Video Resolution	Full HD, 1920x1080 pixels
Lens	NIKKOR glass lens, 5x optical zoom
Focal length	5.0-25.0mm (focal length equivalent to 35mm camera: 28-140mm)
Frames per second	30 fps
Shutter Speed	1/1500 – 1 second
Video file format	MPEG-4 AVC H.264

6.5.2 Set-up of 2D LED tracking system

Both the main camera as well as the camera for redundancy purposes were mounted on tripods, and placed at an oblique angle to the vessel. One was placed alongside the bow and the other alongside the stern. Figure 6.6 and Figure 6.7, show each of the cameras with their respective view angles to the LED rectangle mounted on the vessel.

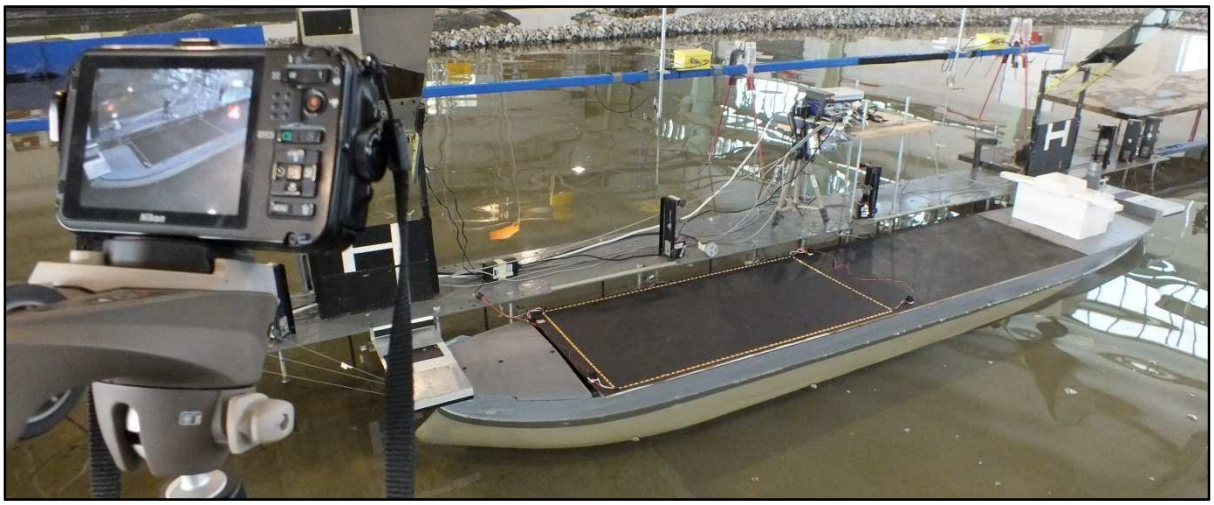

Figure 6.6: Bow camera view of LED rectangle

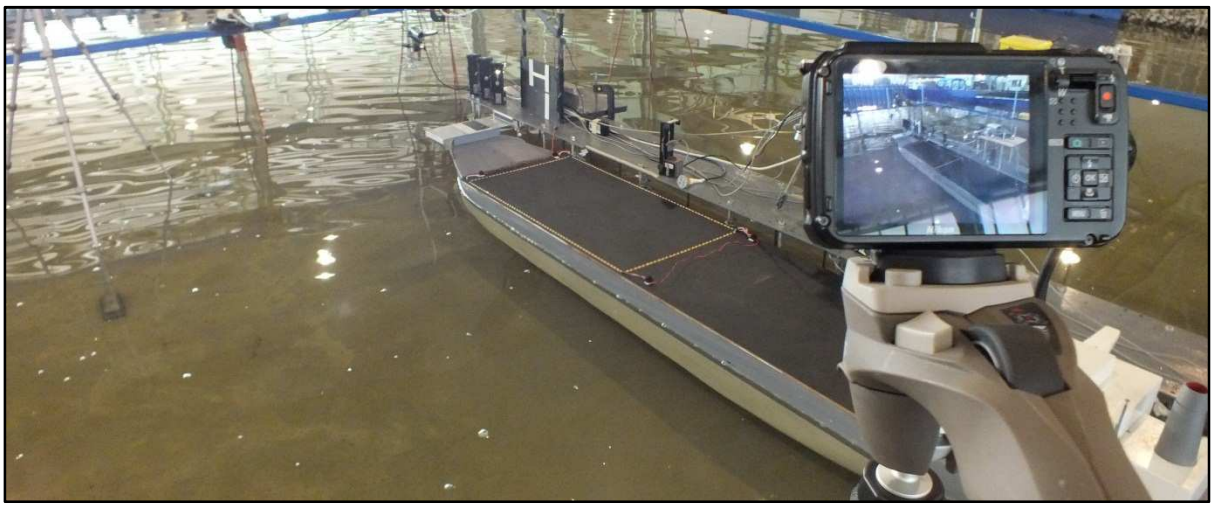

Figure 6.7: Stern camera view of LED rectangle

Automatic focusing and exposure compensation was switched off for both cameras during the set-up procedure. Manual exposure compensation was also set as low as possible for both cameras (with EV = -2). This was done to minimize the effect ambient light other than the light coming from the LED strips might have on the tracking algorithm. The cameras were set to record video files to a secure digital (SD) memory card.

6.5.3 Set-up of 3D Object tracking system

The cameras used for the object tracking tests, were placed on tripods positioned in the same locations as previously placed during the LED tracking testing. Figure 6.8 and Figure 6.9 show each of the cameras with their respective view angles to the object cubes placed on the vessel.

As for the LED tests, automatic focusing and exposure compensation was switched off for both cameras during the set-up procedure. Manual exposure was however left at normal for both cameras (with EV = 0). The cameras were set to record video files to a CD memory card.

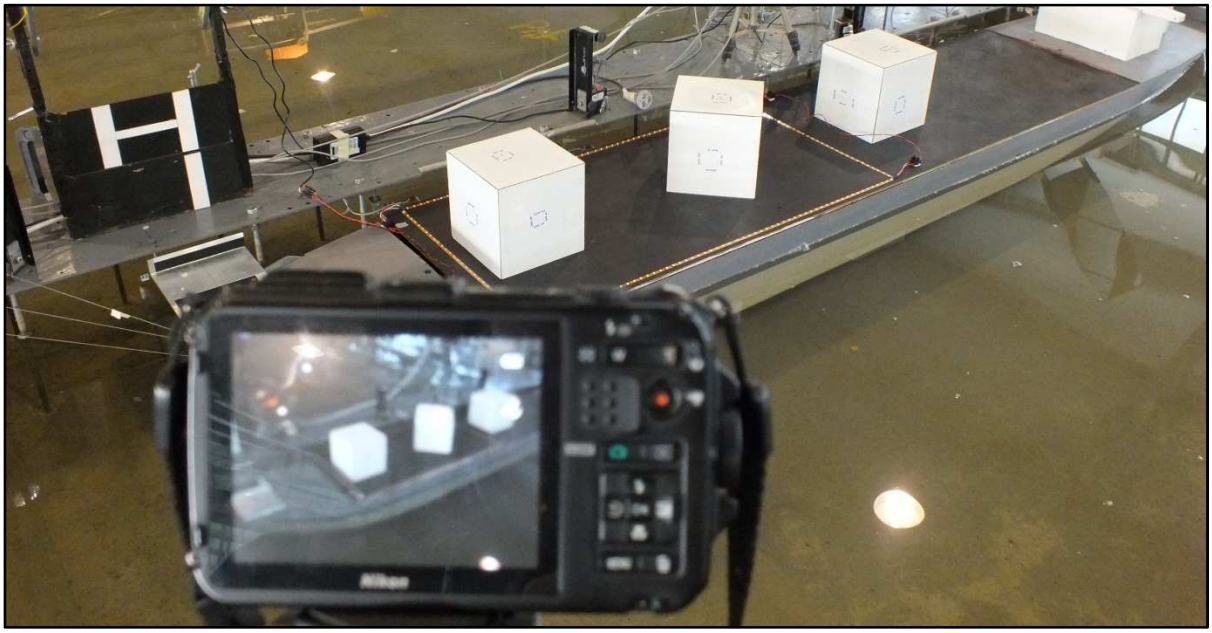

Figure 6.8: Bow camera view of object cubes

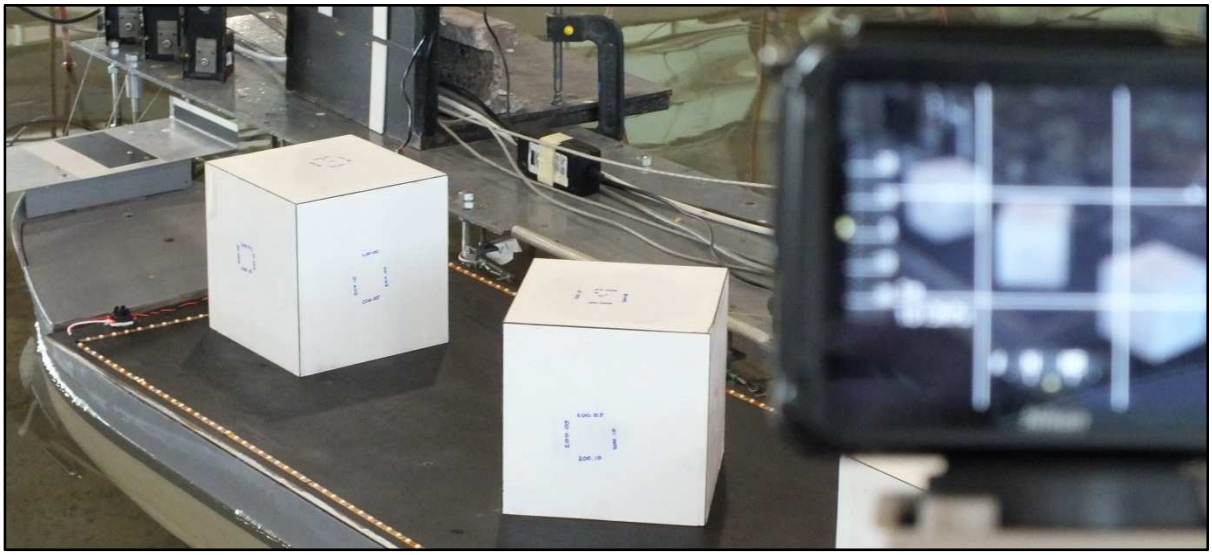

Figure 6.9: Stern camera view of object cubes

6.6 Camera calibration for 2D LED tracking and 3D Object tracking systems

Image processing relies on known camera geometry in order to reconstruct depth when calculating 3D object coordinates from 2D image points. The process used to individually calibrate each camera, using a camera calibration toolbox developed by Jean Yves Bouguet for Matlab is explained in this section. The calibration toolbox is accepted as a recognised camera calibration toolbox for Matlab by the computer vision community and promoted by the California Institute of Technology (also known as Caltech).

The Matlab camera calibration toolbox required images taken of a checkerboard. The checkerboard was orientated differently for each image taken. The images taken were at various distances from the lens to ensure correct calibration over a range of distances. A set of twenty five calibration images were taken with each camera for calibration purposes. Figure 6.10 shows an example of images used in calibration for one of the cameras.

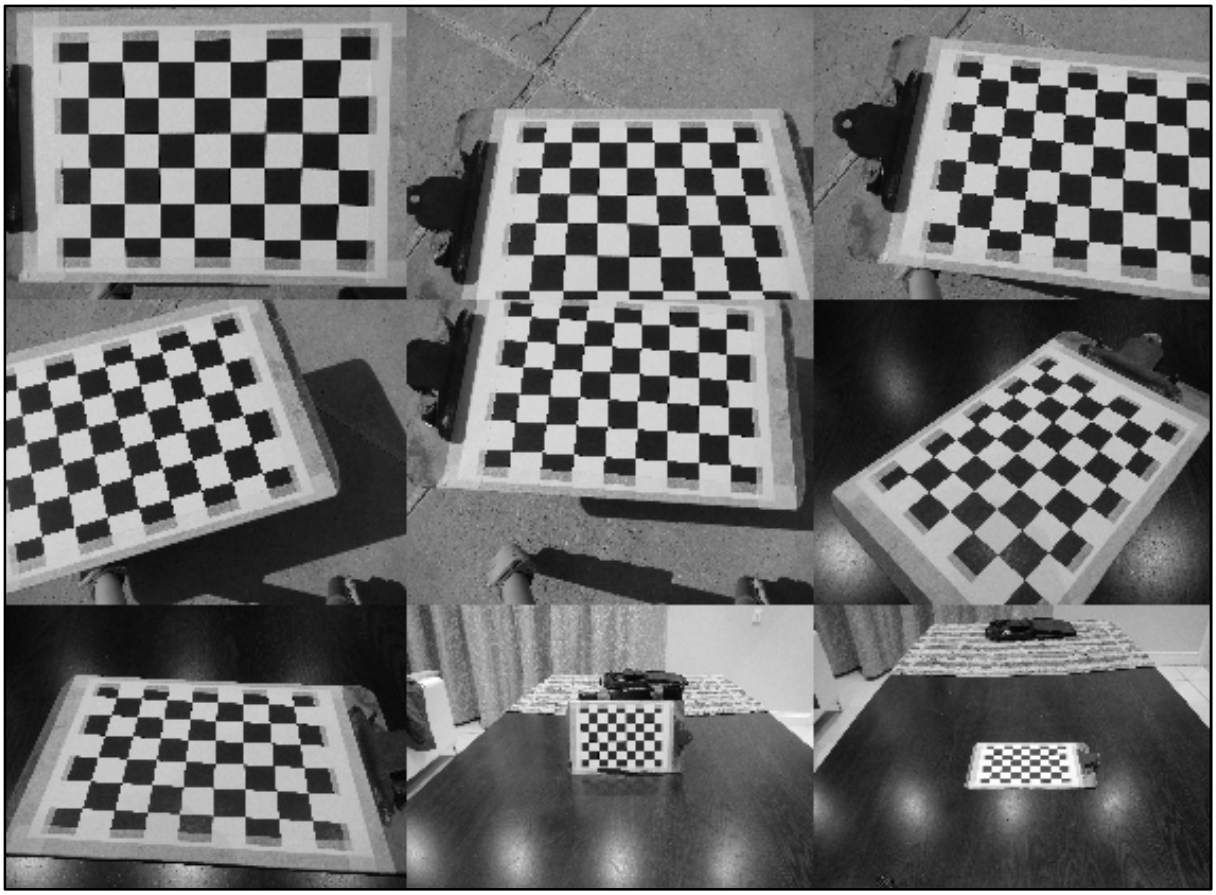

Figure 6.10: Example of checkerboard images taken during calibration

Each camera was calibrated separately with its own set of 25 images. After opening the Matlab camera calibration toolbox, all of the images for each individual camera were read into the toolbox. Figure 6.11 shows the front GUI of the toolbox.

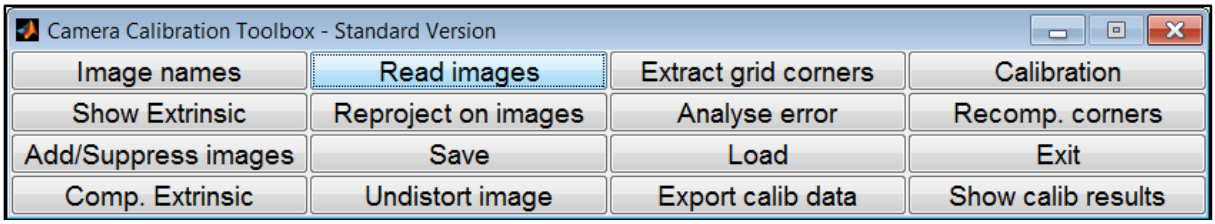

Figure 6.11: Matlab camera calibration toolbox GUI (Yves Bouguet n.d.)

6.6.1 Extracting Grid Corners

After reading in the images, a manual process of extracting corners for each checkerboard image was followed. The process uses the corner extraction engine provided by the toolbox. For each image, the outer corners of the checkerboard were selected. Figure 6.12 shows an example of this process.

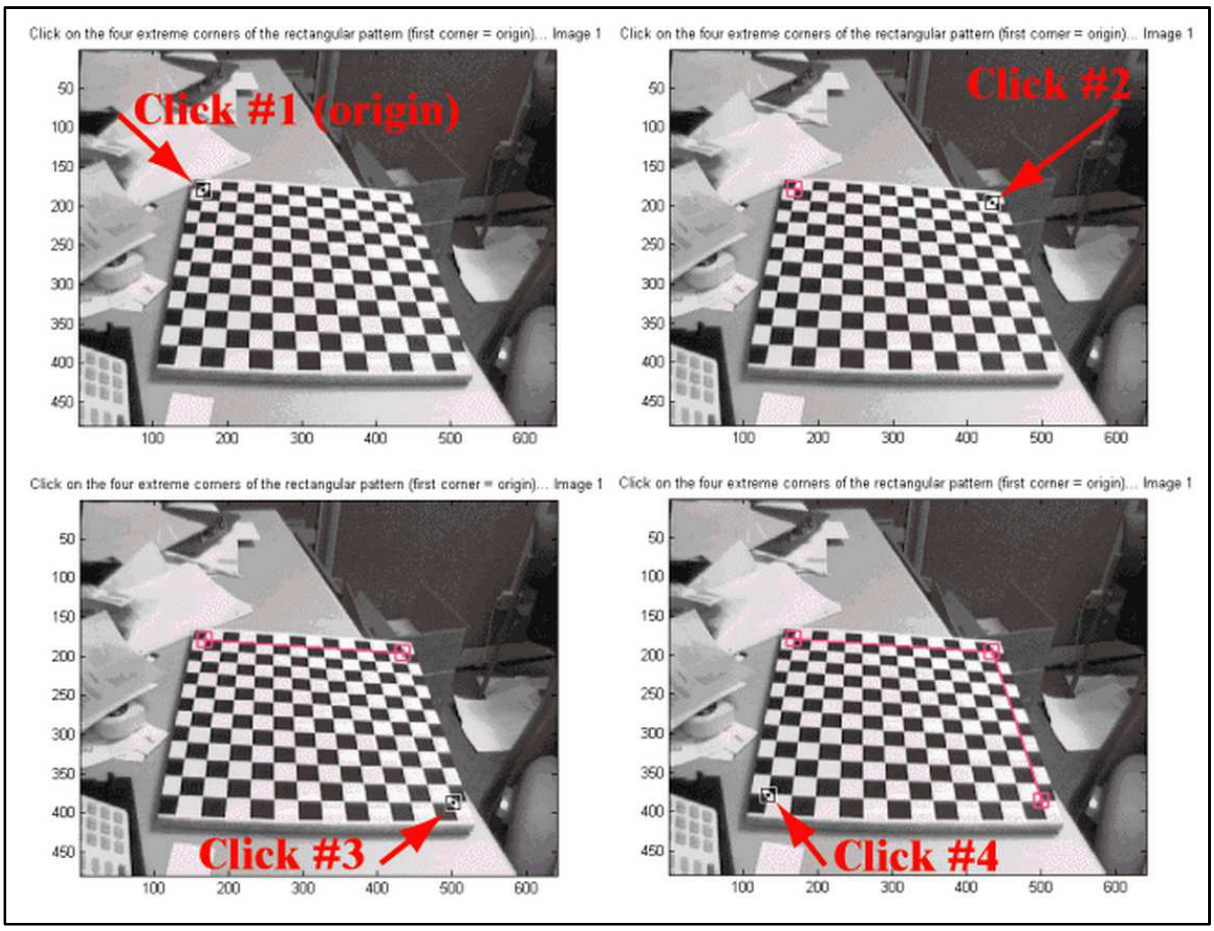

Figure 6.12: Example Image showing the corner extraction manual clicking process (Yves Bouguet n.d.)

The size of each square (30 x 30 mm) on the checkerboard was also provided as input to the corner extraction engine. The corner extraction engine then automatically counted the number of squares in both dimensions, and showed the predicted grid corners in absence of distortion. An example of this is shown in Figure 6.13 with the location of the predicted corners marked by red crosses.

The corner extraction engine then finds the correct location of each corner with an accuracy of about 0.1 pixel (Yves Bouguet n.d.). An example image showing the real grid corners is presented in Figure 6.14.

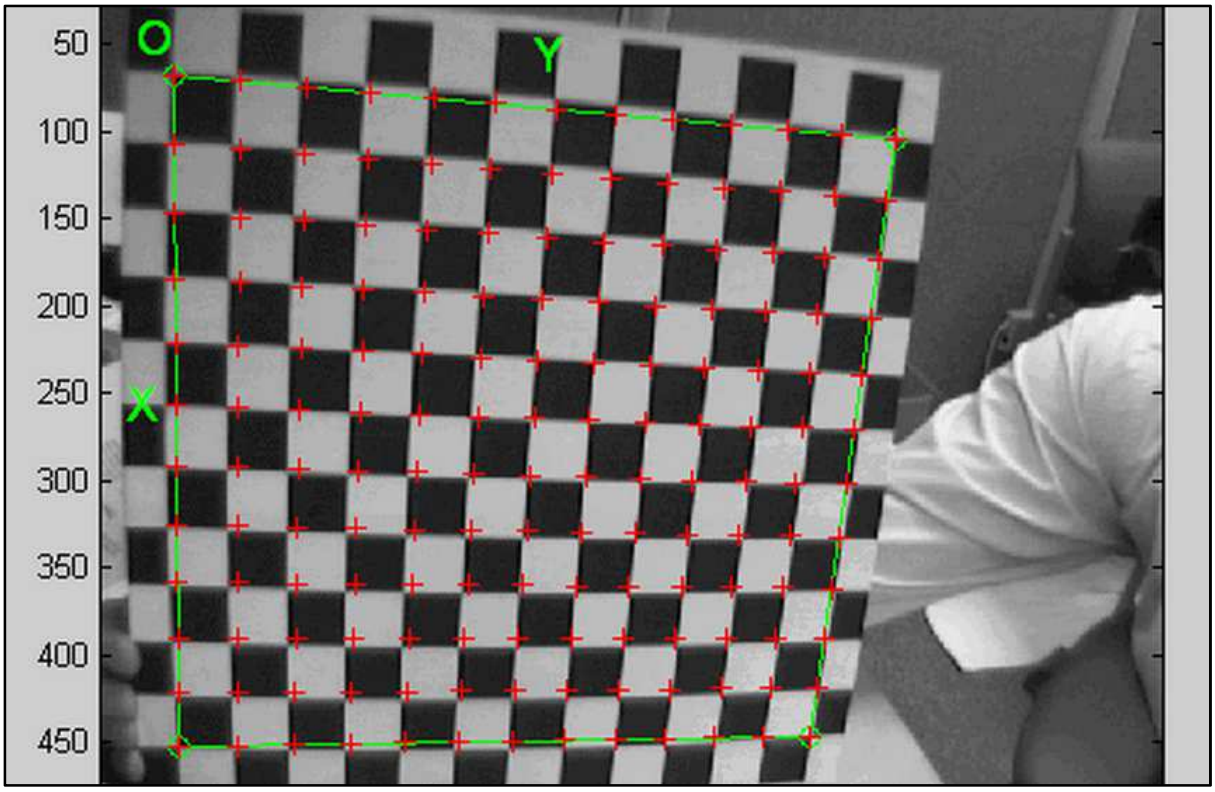

Figure 6.13: Example Image showing the predicted grid corners with red crosses (Yves Bouguet n.d.)

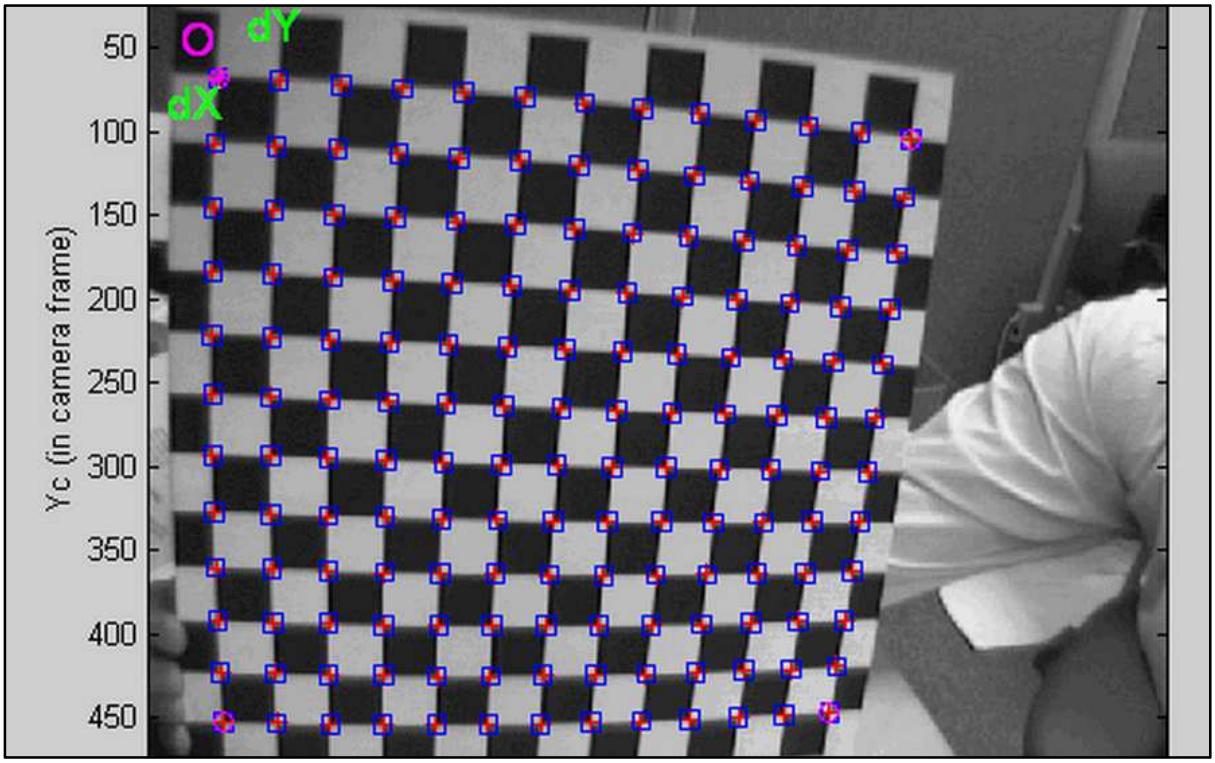

Figure 6.14: Example Image showing the real grid corners with red crosses (Yves Bouguet n.d.)

The corner extraction process was repeated for all 25 images. After corner extraction, a Matlab data file *calib_data.mat* was automatically generated. The file contains all the information gathered throughout the corner extraction process. The information includes image coordinates, corresponding 3D grid coordinates, grid sizes etc.

6.6.2 Camera Parameters

Using the *calib_data.mat* file as input to the calibration engine provided by the toolbox, the extrinsic and intrinsic camera parameters were estimated.

The extrinsic camera parameters define the location and orientation of the camera reference frame with respect to the world reference frame. The intrinsic camera parameters characterize the optical, geometric and digital characteristics of the camera. The intrinsic parameters are necessary to link pixel coordinates of an image point with the corresponding coordinates in the camera reference frame (Bebis 2004).

The estimation is done in two steps namely: initialization and non-linear optimization. The initialization step computes a closed-form solution for the calibration parameters and does not account for lens distortion. The non-linear optimization step minimizes the total re-projection error (in the least square sense) over all the parameters (Yves Bouguet n.d.).

After initialization and optimization, the estimated intrinsic parameters (with uncertainties) were displayed and stored. These parameters include the Focal point, Principal point and Distortion coefficient. An example output of the intrinsic parameters is shown in Figure 6.15.

```
Calibration results after optimization (with uncertainties):

Focal Length:          fc = [ 661.67001   662.82858 ] ± [ 1.17913   1.26567 ]
Principal point:       cc = [ 306.09590   240.78987 ] ± [ 2.38443   2.17481 ]
Skew:             alpha_c = [ 0.00000 ] ± [ 0.00000  ]   => angle of pixel axes = 90.00000 ± 0.00000 degrees
Distortion:            kc = [ -0.26425   0.22645   0.00020   0.00023  0.00000 ] ± [ 0.00934   0.03826   0.00052   0.00053  0.00000 ]
Pixel error:          err = [ 0.45330   0.38916 ]
```

Figure 6.15: Example of intrinsic camera parameters estimated using the Matlab camera calibration toolbox (Yves Bouguet n.d.)

The extrinsic parameters include the general rotation and translation matrices which can be used to convert between the camera and world coordinate systems. The relative 3D pose of the checkerboard with respect to the camera are shown in Figure 6.16. The 3D position and orientation for each checkerboard is independent and has its own extrinsic parameters with respect to the camera.

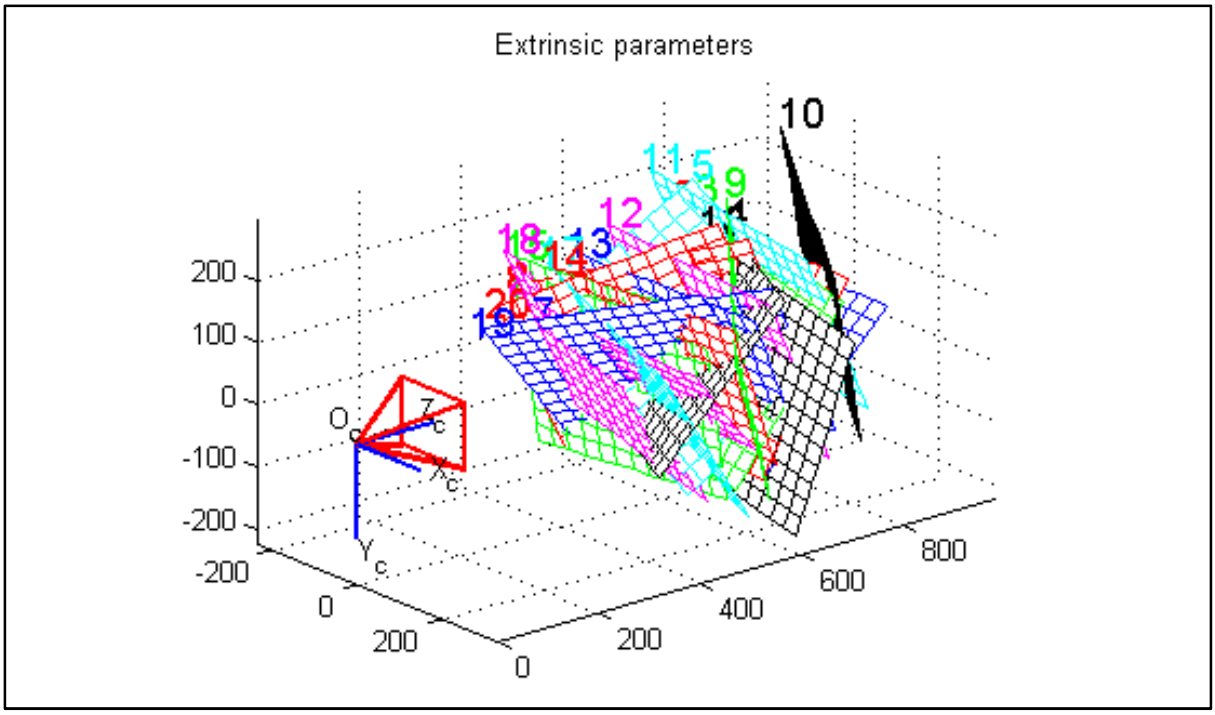

Figure 6.16: Example of relative 3D positions of checkerboard with respect to the camera (Yves Bouguet n.d.)

Only the intrinsic parameters obtained from the camera calibration process will be used in further analysis procedures. The extrinsic parameters describing the 3D pose of the object with respect to the camera will be estimated using a pose estimation algorithm. The pose estimation algorithm will use the intrinsic parameters obtained during calibration as input.

7 PHYSICAL MODEL TEST PROCEDURES AND DATA ANALYSIS

Before the start of a physical modelling test, certain calibration and verification checks are done. These are followed by the starting process, which allows for synchronisation between data acquisition systems. Test preparation procedures, along with all the post processing methods on the raw data captured, are presented in the subsections below.

7.1 Test Preparation and System Synchronization

At the start of each testing day, all the different data acquisition systems along with amplification devices were switched on roughly 45 minutes prior to use. This was to ensure an equilibrium wire temperature in all the wave probe and force measurement cables. A temperature difference in cable wires during instrument calibration or data acquisition could result in incorrect calibration or corrupt data.

Before and after each test, the water level in the basin was checked with a fixed Vernier with accuracy 0.2 mm. The voltage readings of the water level gauges were stored before every test and used as the zero water level mark around which water level elevation oscillated. The recorded zero water level mark can also be used to determine the wave set-up and set-down during a test from the wave height recordings.

The force measurement readings started before the wave generation and continued after the wave generation came to a stop. The measurements taken before and after a test should be equal in value and were to ensure that there was no drift in the measurement values which may point to a sensor incurring some damage during the test.

The cameras used for the developed tracking system were manually started a few minutes before a test. A light in view of the cameras was also switched on at this point. The light was then switched of at the start of a test and used to synchronise the video recordings with all the other data acquisition devices.

A test was started when the oscillations resulting from the calibration and verification activities died down to an extent where the water level was relatively still. The start times of all the different measurement devices for synchronisation purposes were (with the start of a test being at, t = 0 s):

- Start manual recording of developed tracking system (with, t < 0 s)
- Switch on synchronization light (with, t < 0 s)
- Switch off synchronisation light and start data acquisition for Wave gauges, Force measurements and Keoship system (at, t = 0 s)
- Wavemaker started to produce waves (at, t = 60 s)
- Wavemaker stopped (at, t = 960 s)
- Data acquisition for all devices stopped (at, t = 1200 s)
- Stop manual recording of developed tracking system (with, t > 1200 s)

A visual illustration of the start times with respect to measured waves is presented as a time-series in Figure 7.1.

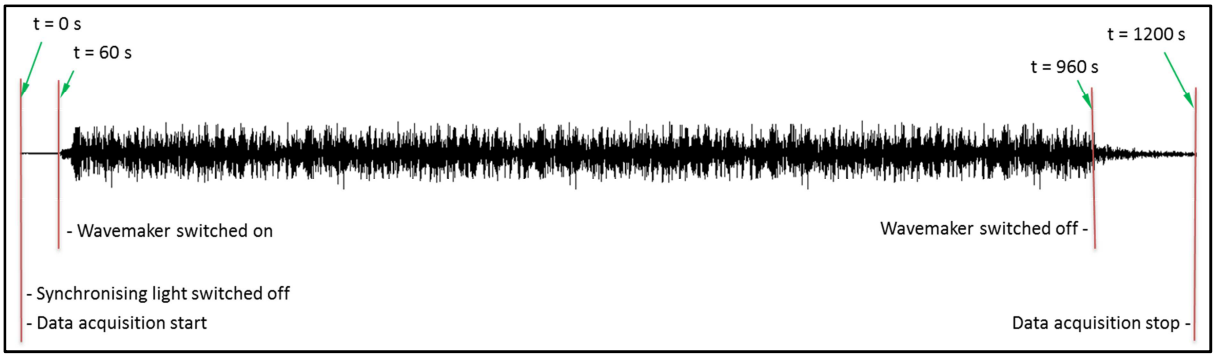

Figure 7.1: Illustration of test procedure in time series form

7.2 Wave Gauge and Keoship Measurement Data Analysis

The results obtained from the wave gauges and Keoship system were analysed using in-house post-processing scripts from the CSIR.

The wave heights and Keoship motions measured are all input as text format to these post processing scripts along with the ship particulars, loading conditions and mooring arrangement information.

The analysis period was chosen to start only 180 s after the wavemakers started and stopped at the same time when the wavemakers were switched off. A visual illustration of the analysis period is presented in Figure 7.2.

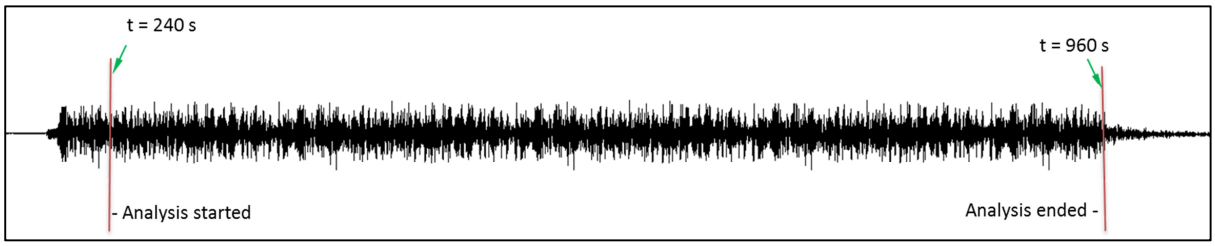

Figure 7.2: Data analysis period

7.3 Analysis of LED and Object Tracking Video

For both of the cameras used, the manual recordings were imported into Microsoft movie maker to manually synchronise with the start time and stop time of each test. Once imported, each recording was manually edited to cut off video information to where the synchronization light is switched off and again, 1200 s after the synchronisation is switched off. The process is accurate to 1 frame while the video recordings had a frequency of 30 fps, resulting in an editing accuracy of roughly 0.03 seconds. The editing process takes roughly 2 hours for editing both video recordings of a test.

The edited video was then imported into Matlab where it was converted into individual image frames which were then saved onto a hard drive. Each recording of 1200 seconds was converted into 36000 individual frames. The converting process takes a further hour per test recording.

After analysing the keoship data for each batch of tests, a section of data will be selected to compare with the tracking algorithm method. Due to the processing time required to run the tracking

algorithm on all 36000 individual frames, only a small number of image frames will be processed and compared to the Keoship motions for the same time period. The section of images to process and compare will overlap with the Keoship analysis section where maximum motions occurred. The portion was selected to include the build-up of energy beforehand, as well as a portion after the maximum motions. Figure 7.3, illustrates Keoship motions at prototype with a typical selected overlap area.

The image processing done on the image frames and the tracking analysis are described in the subsections below.

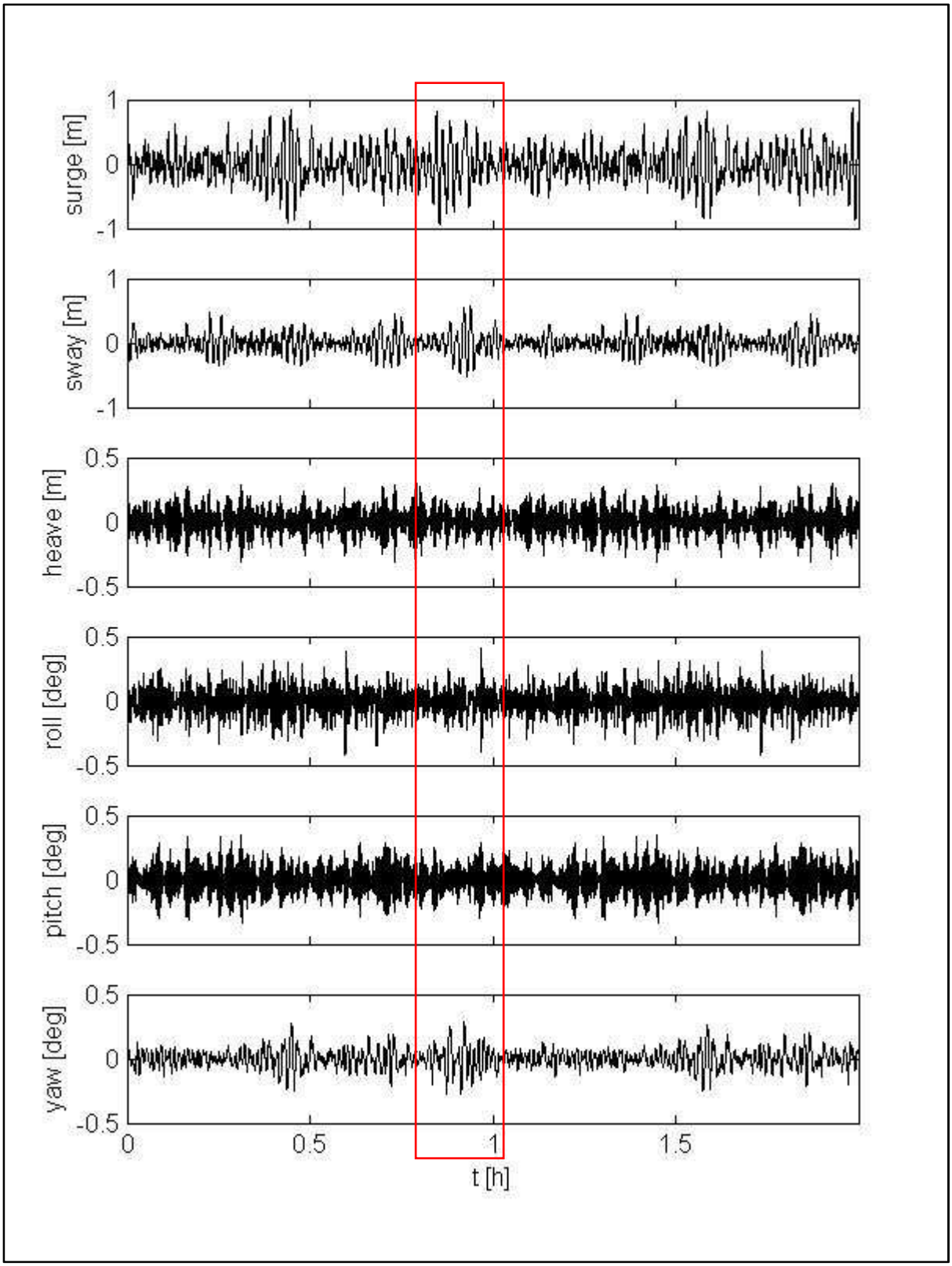

Figure 7.3: Typical Keoship motion results showing the section within the red block chosen as the comparison area for the tracking algorithm results.

7.3.1 Image processing for LED tracking

The author developed an image processing script in Matlab which individually processes each image frame of the tracking video.

The Matlab script reads in one image frame at a time and converts the truecolor image to a grayscale intensity image. The greyscale image is then cropped in an effort to remove the majority of unnecessary information around the object to be tracked. This also removes image portions around the object which may include reflection light on the water surface and then hinder the tracking algorithm. An example of an image before being cropped, showing water reflections around the LED object is presented in Figure 7.4.

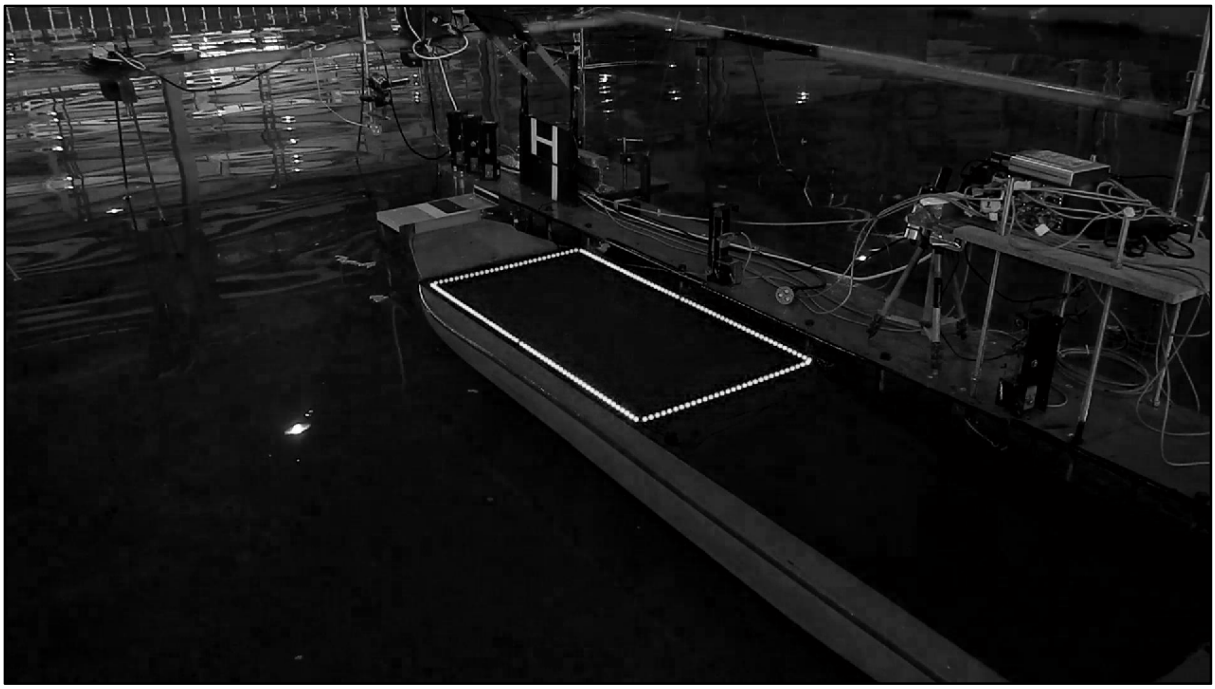

Figure 7.4: Example image showing water reflections on certain image portions around the LED object

The cropped grayscale image is then converted to a black and white image (binary image). The binary image is then eroded once, and dilated three times. “Erode” (which means to “shrink”), in effect reduces the white spaces (binary value = 1) by converting any white pixels directly adjacent to black pixels (binary value = 0) to black. Dilating is exactly the opposite of erode, and in effect expand the white spaces. The eroding and dilating process is illustrated in Figure 7.5.

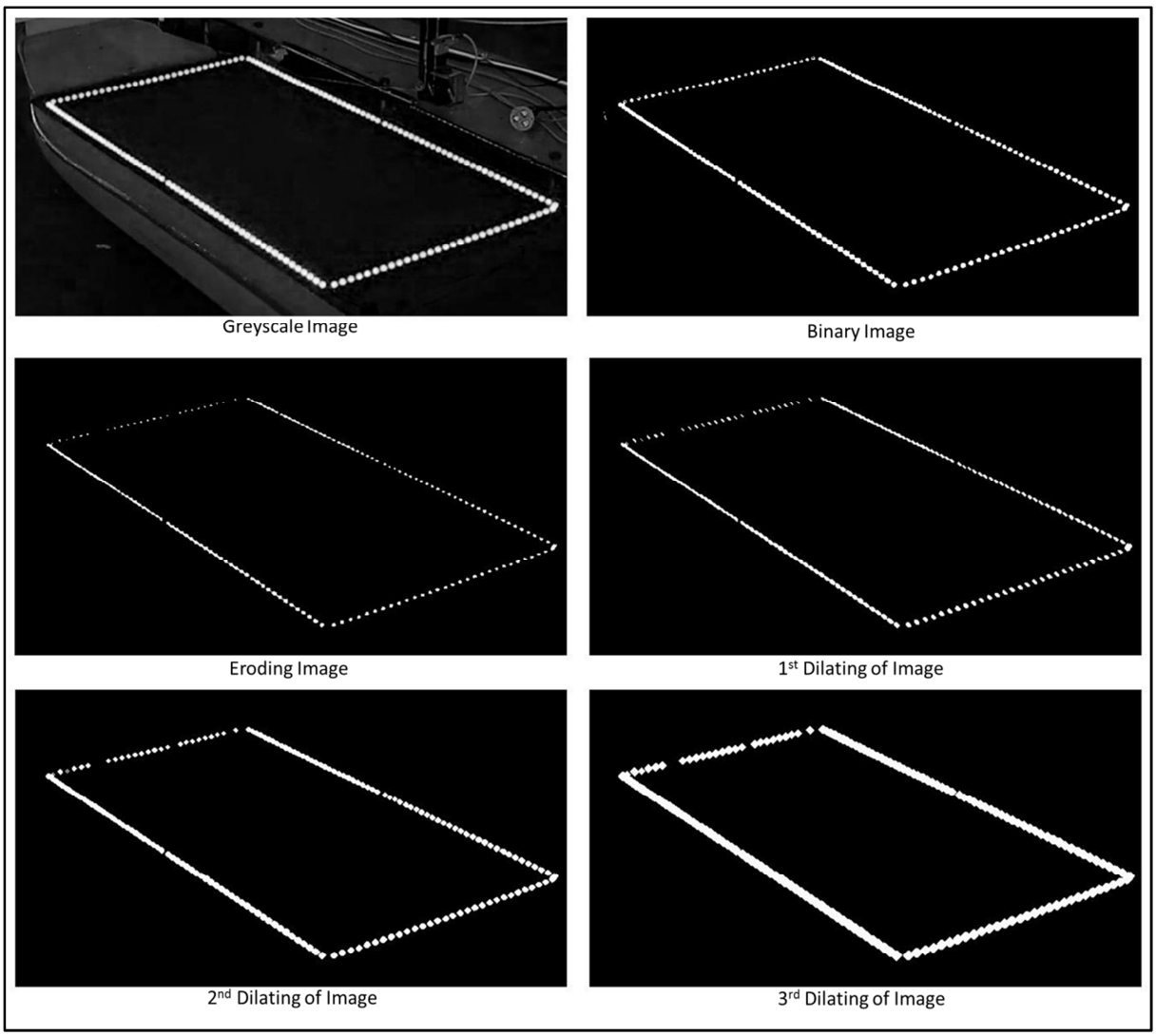

Figure 7.5: Illustration of dilating and eroding process

After dilation, the pixel values are multiplied by 255. This results in white pixels, changing its binary value from 1 to 255. Regional maxima filtering using 8-connected neighbouring pixels are then run on the image. The pixel location of these maximum values (i.e. LED locations with binary pixel values = 255) are then stored in vector form. The pixel coordinates for all four corners of the LED square are then calculated and exported to a text file. This process uses FOR and IF statements to compare the binary value of each pixel to its neighbour. An example image showing the maximum values obtained through filtering in blue along with the calculated corner locations is presented in Figure 7.6. The Matlab image processing script for the 2D LED tracking is provided in Appendix A for reference.

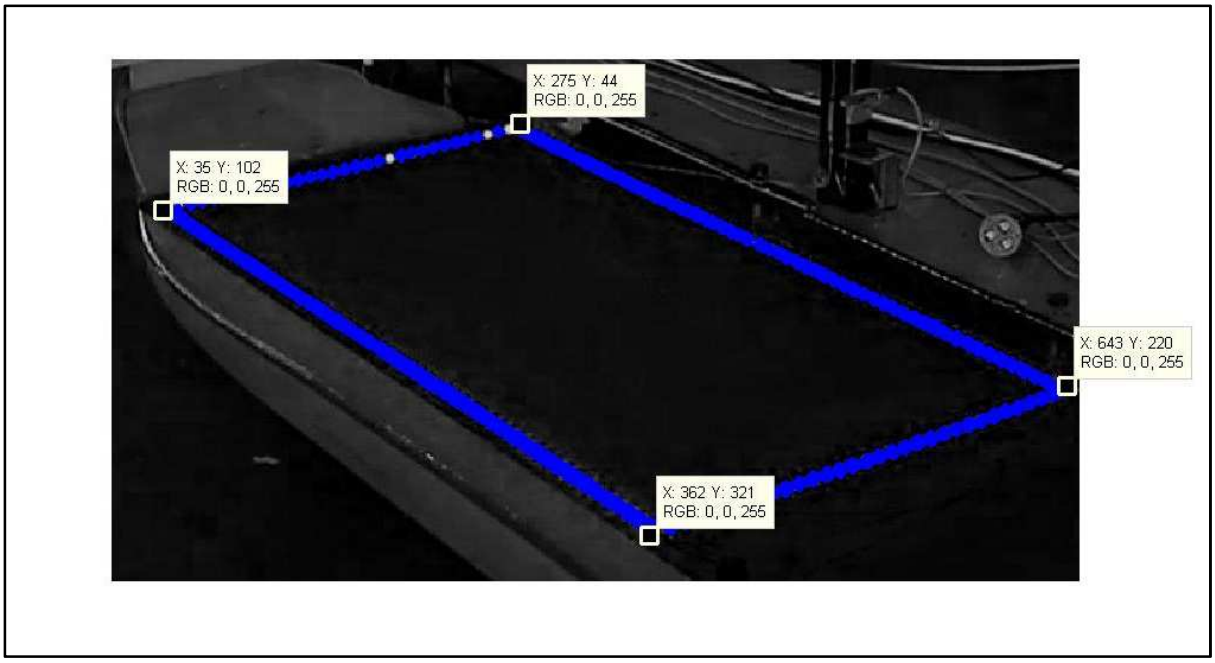

Figure 7.6: Illustration of extracted corner locations of LED square

7.3.2 3D pose estimation for LED tracking

After obtaining the image coordinate points for each individual frame, a 3D coplanar pose estimation application by Kirillov (2011), was used to estimate the pose for each individual frame. The application implements a coplanar Pose from Orthography and Scaling with ITerations (POSIT) algorithm to obtain the 3D pose of the object.

The application, written in C Sharp programing language, originally only allowed for the image points of a single image to be manually typed in the GUI. The model coordinates (mm) of the image points were originally also typed in the GUI for scaling purposes.

The application by Kirillov (2011), was altered by the author to take as input the output coordinates of the points from the image processing script. The model coordinates of the corresponding image coordinates are also provided as input, along with the intrinsic camera parameters obtained from the camera calibration process.

The rotation and translation matrices for each image frame are written to a .txt file. The rotation and translation matrices are the extrinsic parameters describing the 3D pose of the object from the camera perspective. The altered C sharp code is provided in Appendix B for reference.

7.3.3 Image processing for Object tracking

The 3D Object tracking system was developed by the author. During the development of the proof-of-concept 3D Object tracking algorithm, it was decided not to go the wire-frame route in the first concept development. Rather, to simplify matters, it was decided to use general image processing to locate the object corners, which will then be input to a pose estimation algorithm.

As for the LED image processing, the Matlab script reads in one image at a time, and converts the truecolor image to a grayscale intensity image. For the tests, it was decided to track the block which is the closest to the camera. The greyscale image is then cropped in an effort to remove the majority of unnecessary information around the object to be tracked. Such as light reflection on the water surface which would hinder the tracking algorithm. An example of an image before being cropped, showing water reflections around the LED object, is presented in Figure 7.7.

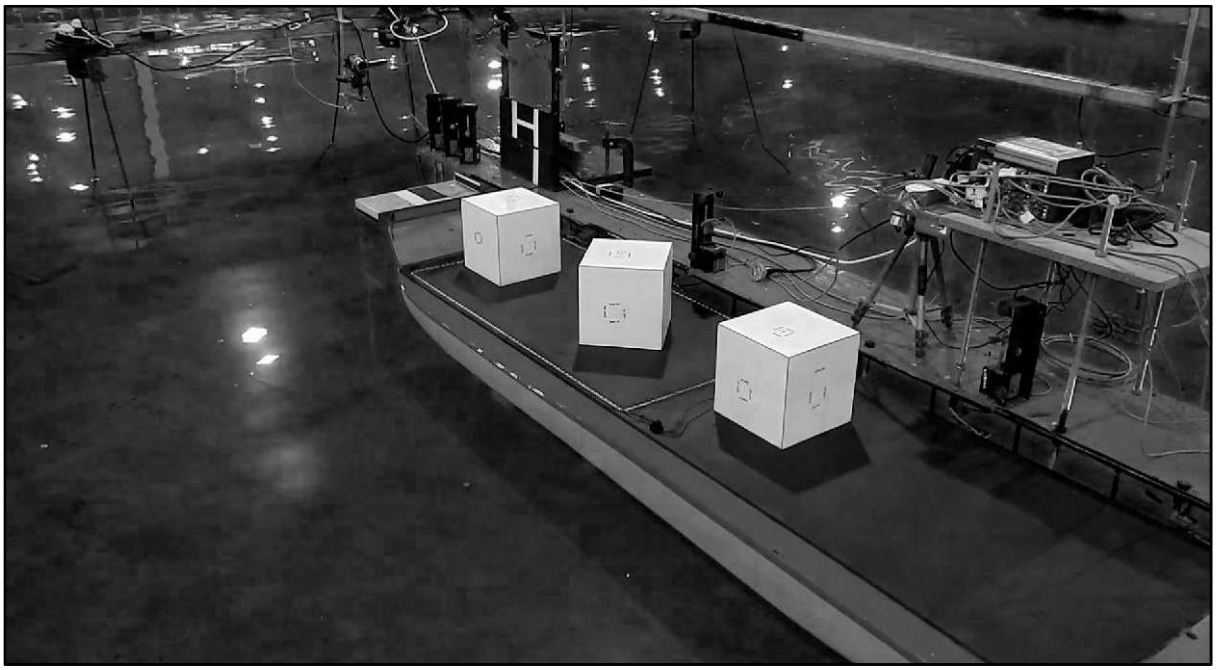

Figure 7.7: Example image showing water reflections on certain image portions around the cube objects

The cropped grayscale image is then converted to a black and white image (binary image). The binary image is then eroded and dilated once. An example of an image after being cropped, eroded and dilated is preesented in Figure 7.8.

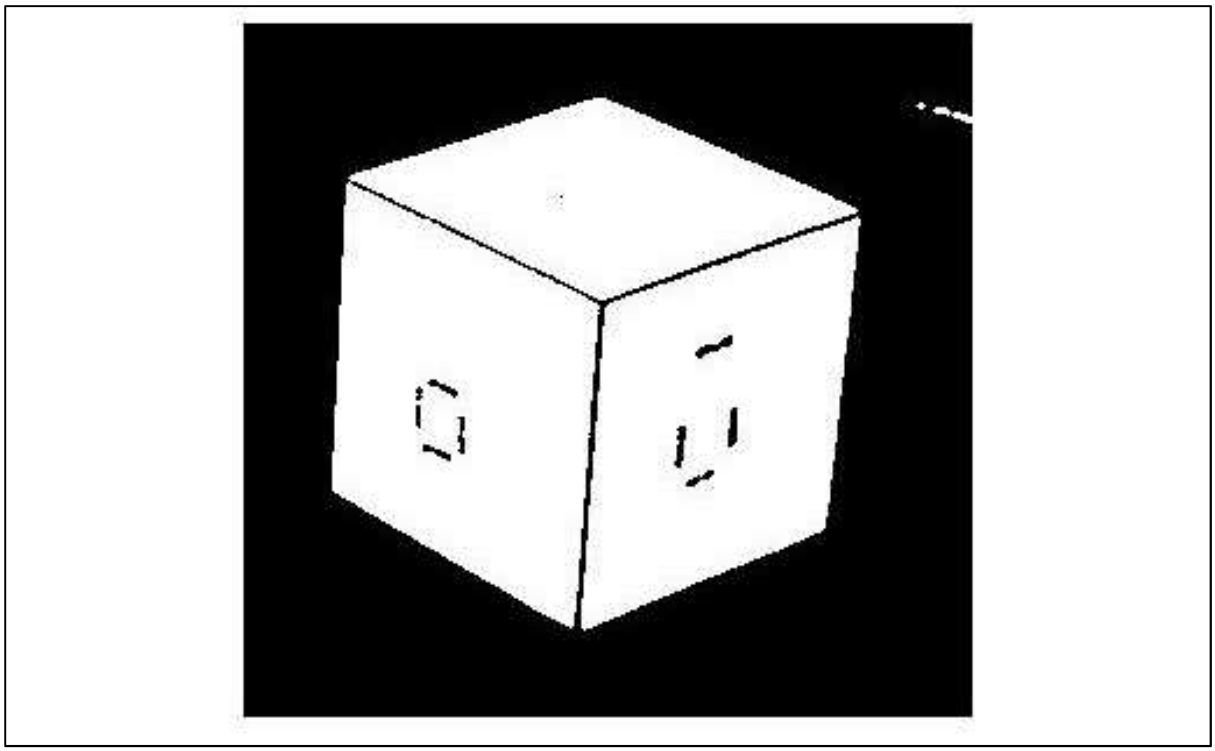

Figure 7.8: Example image showing cropped area with cube object to be tracked

After dilation, the pixel values are multiplied by 255. This results in white pixels, changing its binary value from 1 to 255. The connected components of the block are then separated into its individual 3 individually labelled parts, as shown in Figure 7.9.

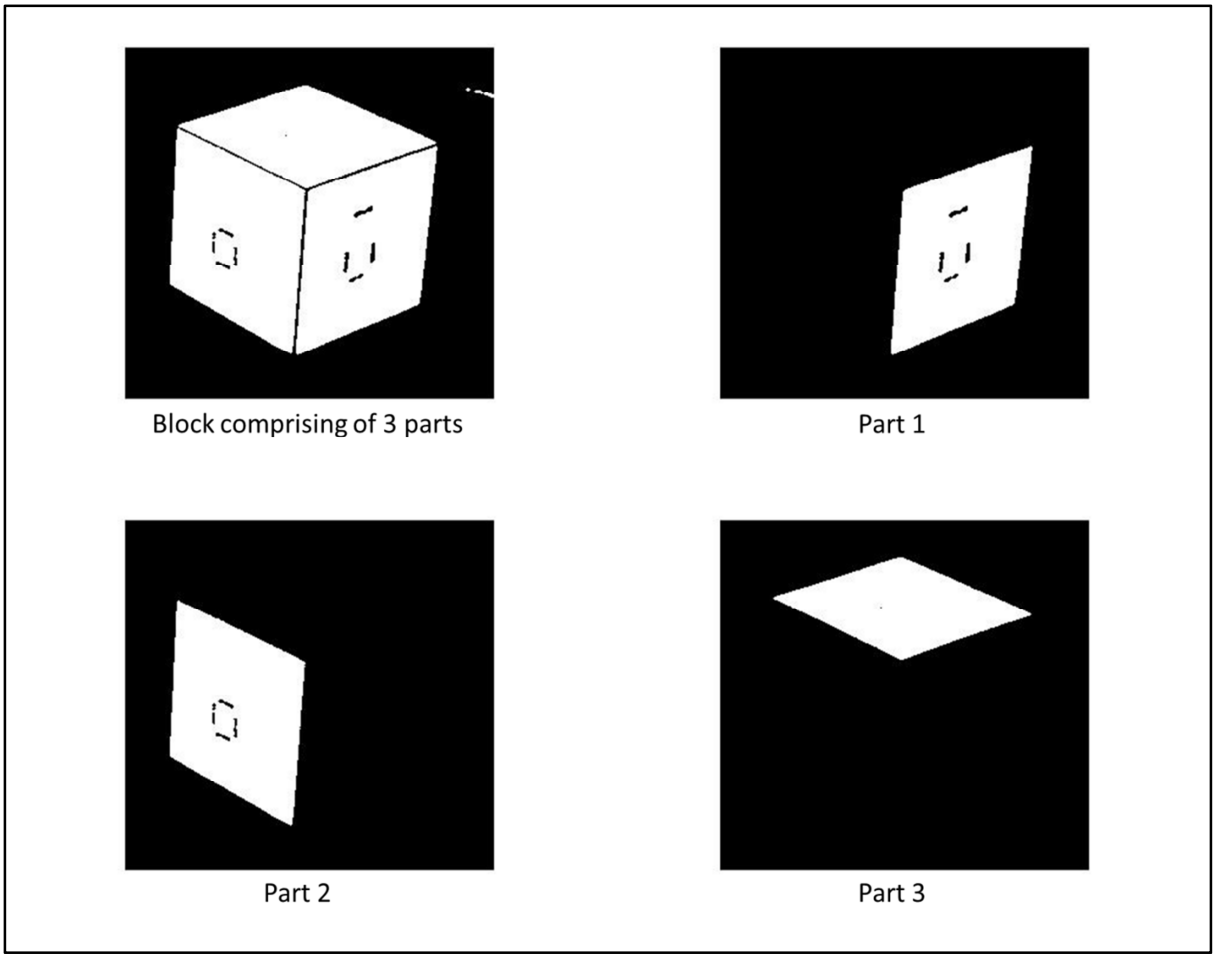

Figure 7.9: Illustration on connected components being separated

As for the LED image processing, Regional maxima filtering using 8-connected neighbouring pixels, are then run on each of the parts. The pixel coordinates for all four corners of each separated part are then calculated. This process uses FOR and IF statements to compare the binary value of each pixel to its neighbour. The individual corner calculations of each part are then combined to export the visible object corners of the cube object to a .txt file. The Matlab image processing script for the 3D Object tracking is provided in Appendix C for reference.

7.3.4 3D pose estimation for Object tracking

For the 3D pose estimation, the image coordinates are used as well as the object coordinates of the same points and the intrinsic camera parameters. Unlike the Coplanar POSIT algorithm used for the LED tracking, model points are not on the same plane.

To calculate 3D pose for the non-coplanar points, a Matlab script, written by De Menthon (2003), was slightly adapted by the author and used to calculate the rotation and translation matrices for each image frame. The altered Matlab script is provided in Appendix D for reference.

7.3.5 Converting rotation and translation matrices to scaled 6DOF from the camera perspective

This part of the analysis converts the camera perspective rotations and translations to scaled rotations and translations around the vessels CoG.

For object rotations, the rotation matrices from a camera perspective are rotated to an object perspective with the rotation of the first frame taken as the zero starting point. L Terblanche and W Brink (2014, pers.comm.,26 September) from the applied mathematics department at the University of Stellenbosch, suggested that converting from camera perspective to object perspective, is as simple as multiplying all rotating and translating information with the transpose rotation of the zero position seen from the camera position.

Using this as starting point, and working with examples of known rotation and translation, revealed that the zero point rotation is simply the transpose of the first frame's rotation matrix, multiplied by the rotation matrix of the first frame and can be mathematically written as:

$$R_0' = R_0^T \times R_0 \tag{7.1}$$

With the rotation of the first frame taken as the zero point, every sequential rotation matrix from an object perspective can be mathematically presented as (where 'i' denotes the frame number):

$$R_i' = R_0^T \times R_i \tag{7.2}$$

Roll, pitch and yaw values (in degrees) from an object perspective are then calculated using a Matlab program script, 'tr2rpy.m' by Corke (2011). The script converts the rotation matrices, R_i' to the respective roll, pitch and yaw rotations in degrees.

For object translations, the difference between each translation matrix and the zero starting point is calculated and then rotated from a camera perspective to an object perspective. The translation matrices between image frames are simply the difference between sequential translation matrices:

$$T_i - T_{i-1} \tag{7.3}$$

Rewriting the above to be the difference between the first frame taken as the zero starting point and sequential frames it becomes:

$$T_i - T_0 \tag{7.4}$$

The above are however only translation differences seen from the camera perspective. The translation differences are then rotated to be from the object perspective. This is done by multiplying the translation differences with the transpose of the original rotation matrix and the new object perspective rotation matrix (i.e. $R_i' R_i^T$). The translation differences from the object perspective can then be mathematically written as:

$$T_i' = R_i' \times R_i^T \times (T_i - T_0) = \begin{bmatrix} x_i \\ y_i \\ z_i \end{bmatrix}$$

(7.5)

Where x_i, y_i and z_i refer to the surge, sway and yaw around the zero location of the object.

Converting the calculated translations from the object bound coordinate system to the CoG of the vessel, the inverse of equation (2.5) is applied.

$$\begin{aligned} x_{COG} &= x_i + y_b\psi - z_b\theta \\ y_{COG} &= y_i - x_b\psi + z_b\varphi \\ z_{COG} &= z_i + x_b\theta - y_b\varphi \end{aligned} \tag{7.6}$$

Where x, y and z are the vessel motions surge, sway and heave respectively, while x_b, y_b and z_b are the body bound coordinates of object centre on the vessel and the roll, pitch and yaw calculated by Corke (2011) are φ, θ and ψ repectively.

The translation and rotations are then scaled to prototype values before being written to a .fig file which is then compared to the motion results from the Keoship system. The Matlab program scripts used for all the calculations above is presented in Appendix D and Appendix E for reference.

8 PHYSICAL MODEL TEST RESULTS

The physical model results for the 2D LED tracking tests and 3D Object tracking are provided in the subsections below. The results are compared to that obtained with the Keoship system as well as to consecutive repeatability tests with the same wave condition. The root-mean-square error, R_{rms} is also used to quantify the difference between the data sets. For comparison with the Keoship system, all data are presented in prototype, except for the, R_{rms} which is presented in model scale to best quantify the difference. The 6DOF motion plots for the 2D LED tracking system tests and 3D object tracking tests are provided in Appendix G.

Results of the physical model testing show that there are gain differences between the measured translation amplitudes for both the tracking systems when compared to the Keoship system. In order to visually compare data trends and shape, the amplitudes of the 2D LED tracking data sets were divided by a factor of five for the comparison plots. The gain differences are discussed further in the discussion section.

8.1 Results from Keoship System

The Keoship motion results were fully analysed for all the 2D LED and 3D Object tracking tests. Analysis showed good repeatability for all four test conditions; for both the 2D LED tracking and 3D Object tracking tests. The repeatability is illustrated by comparing Test LED01 and Test LED02 in Figure 8.1.

For Test LED01 the surge, sway, heave, roll, pitch and yaw measured a maximum of 0.8 m, 0.6 m, 0.25 m, 0.39°, 0.17° and 0.33° in prototype respectively. The accuracy differences between the two repeatability tests are quantified with the root-mean-square error between Test LED01 and Test LED02 in Table 8.1.

Table 8.1: Accuracy difference between Keoship motions for test LED01 and LED02

Motion	R_{rms} (model scale)
Surge [mm]	0.53
Sway [mm]	0.27
Heave [mm]	0.56
Roll [°]	0.04
Pitch [°]	0.04
Yaw [°]	0.02

The R_{rms} values show that test results are repeatable with measurement differences smaller than 0.6 mm for translation and 0.04° for rotation.

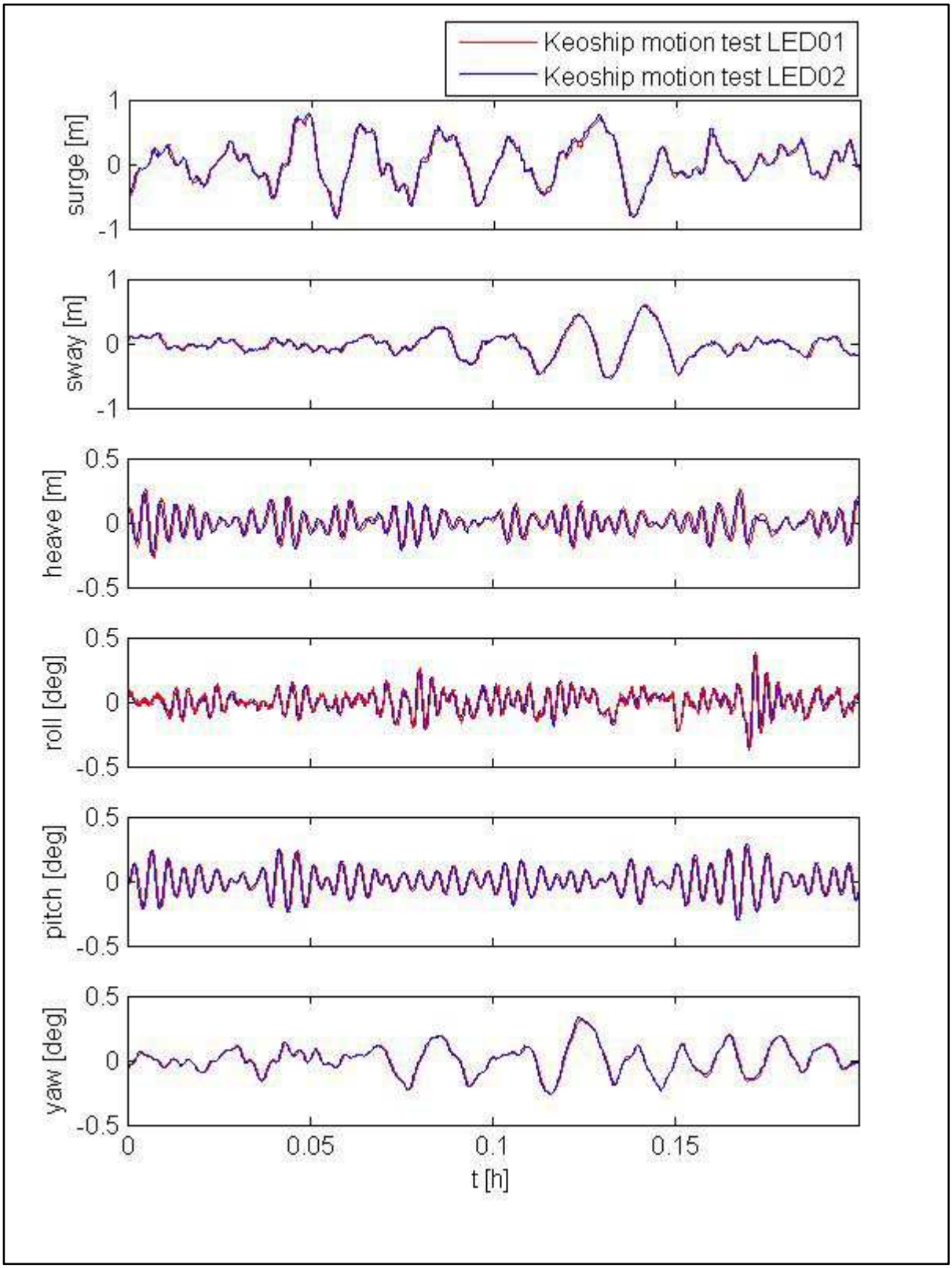

Figure 8.1: Keoship system repeatability for Test LED01 and LED02

8.2 Comparison between 2D LED Tracking and Keoship method

All of the 2D LED measurement data exhibited a lot of unwanted transients, or spikes in the data that needed to be discarded. All data points falling outside the maximum expected motions between image frames were removed. With the expected translation being less than 10 mm and rotation less than 1.0° between images (with a frame rate of 30 fps). The data was also filtered with a low pass filter in an effort to get rid of all the high frequency noise. The effect of the transient points and high frequency noise is illustrated in Appendix F with Test LED02 as example.

To accurately quantify the differences between the data sets, the root-mean-square error values are calculated before this gain adjustment and therefore appear different to what is shown in the comparison plots.

8.2.1 Wave condition (H_{m0} of 1.5 m and T_P of 16 seconds prototype)

Using test LED02, the accuracy difference between the 2D LED tracking system and Keoship method are quantified in Table 8.2, and graphically shown in Figure 8.2. The accuracy difference between the three repeatability Tests LED01, LED02 and LED03 are quantified in Table 8.3, and graphically shown in Figure 8.3.

Table 8.2: Accuracy difference between 2D LED tracking and Keoship system, using Test LED02

Motion	R_{rms} (model scale)
Surge [mm]	9.62
Sway [mm]	11.60
Heave [mm]	7.63
Roll [°]	0.17
Pitch [°]	0.17
Yaw [°]	0.11

Table 8.3: Accuracy difference between 2D LED tracking tests for LED01, LED02 and LED03

Motion	R_{rms} (model scale) for LED01 and LED02	R_{rms} (model scale) for LED01 and LED03	R_{rms} (model scale) for LED02 and LED03
Surge [mm]	11.00	9.05	7.00
Sway [mm]	7.15	6.30	4.20
Heave [mm]	6.65	6.25	6.20
Roll [°]	0.20	0.18	0.09
Pitch [°]	0.24	0.24	0.01
Yaw [°]	0.26	0.25	0.06

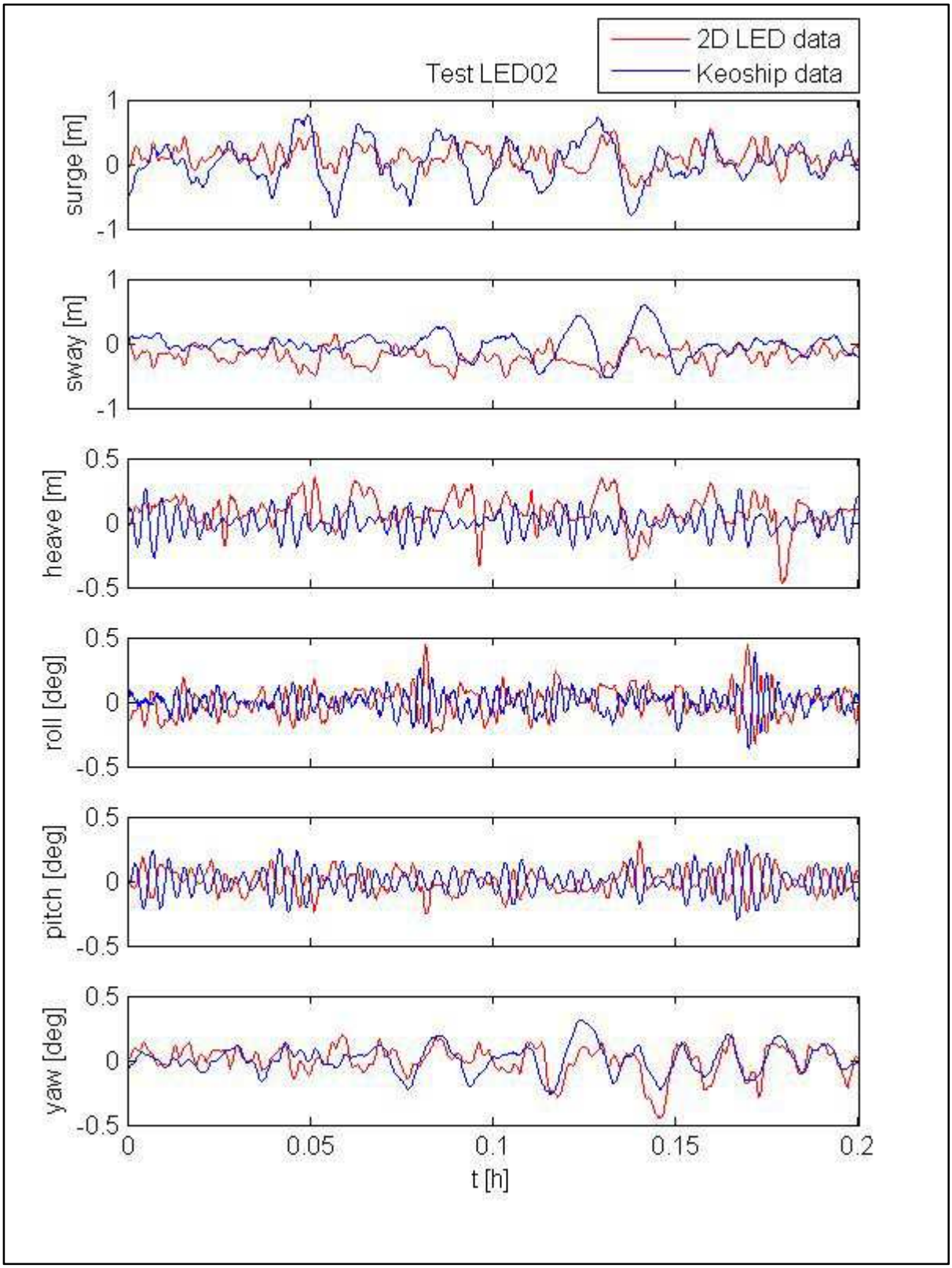

Figure 8.2: Data comparison for test LED02

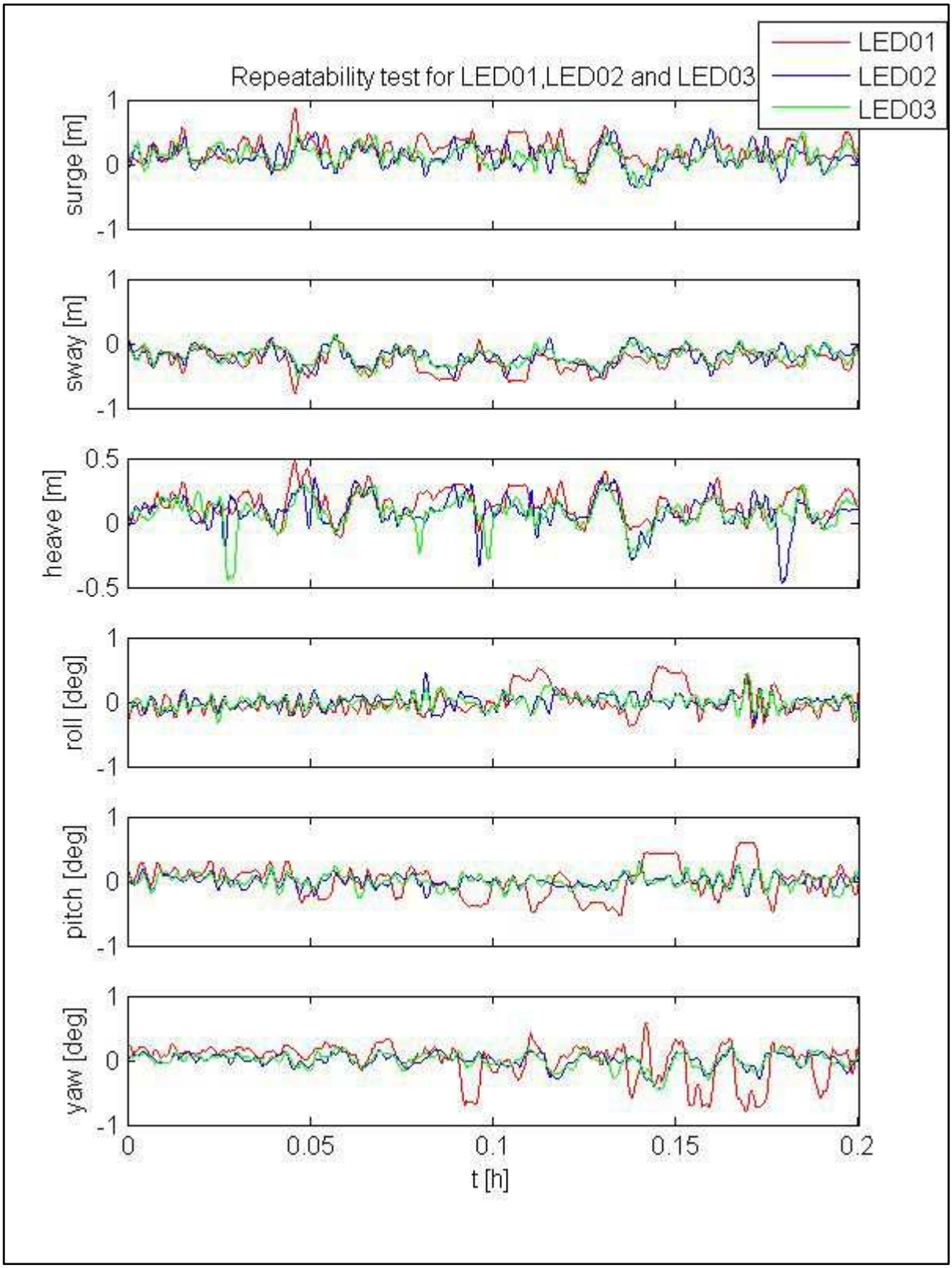

Figure 8.3: Repeatability plots for tests LED01, LED02 and LED03

8.2.2 Wave condition (H_{m0} of 3.0 m and T_P of 16 seconds prototype)

Using test LED06, the accuracy difference between the 2D LED tracking system and Keoship method are quantified in Table 8.4, and graphically shown in Figure 8.4. The accuracy difference between the three repeatability Tests LED04, LED05 and LED06 are quantified in Table 8.5, and graphically shown in Figure 8.5.

Table 8.4: Accuracy difference between 2D LED tracking and Keoship system, using Test LED06

Motion	R_{rms} (model scale)
Surge [mm]	14.84
Sway [mm]	20.38
Heave [mm]	13.60
Roll [°]	0.30
Pitch [°]	0.31
Yaw [°]	0.20

Table 8.5: Accuracy difference between 2D LED tracking tests for LED04, LED05 and LED06

Motion	R_{rms} (model scale) for LED04 and LED05	R_{rms} (model scale) for LED04 and LED06	R_{rms} (model scale) for LED05 and LED06
Surge [mm]	43.10	43.05	13.80
Sway [mm]	17.15	17.70	14.10
Heave [mm]	12.65	13.65	8.90
Roll [°]	1.72	1.72	0.21
Pitch [°]	0.26	0.26	0.16
Yaw [°]	0.21	0.25	0.16

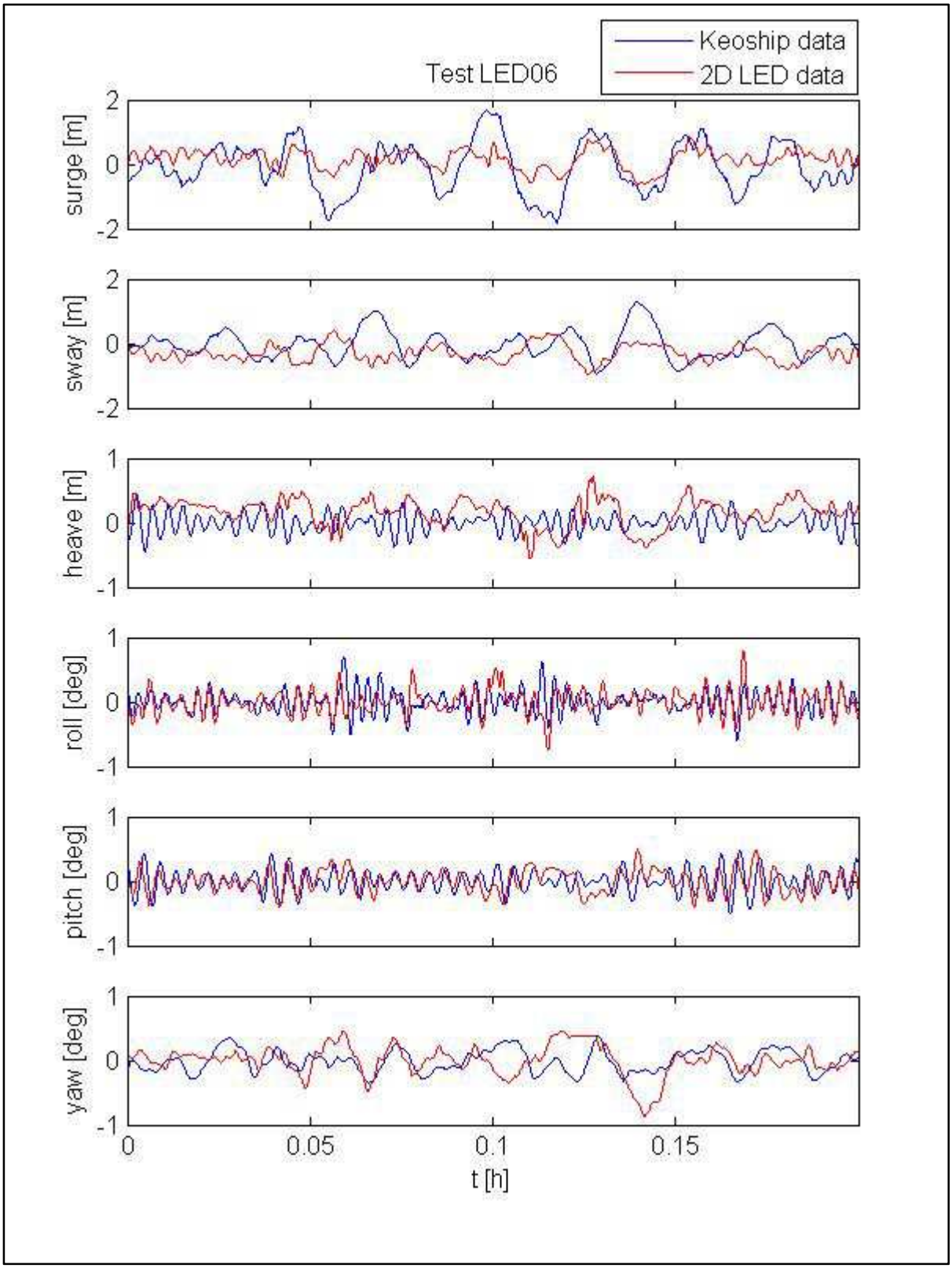

Figure 8.4: Data comparison for test LED06

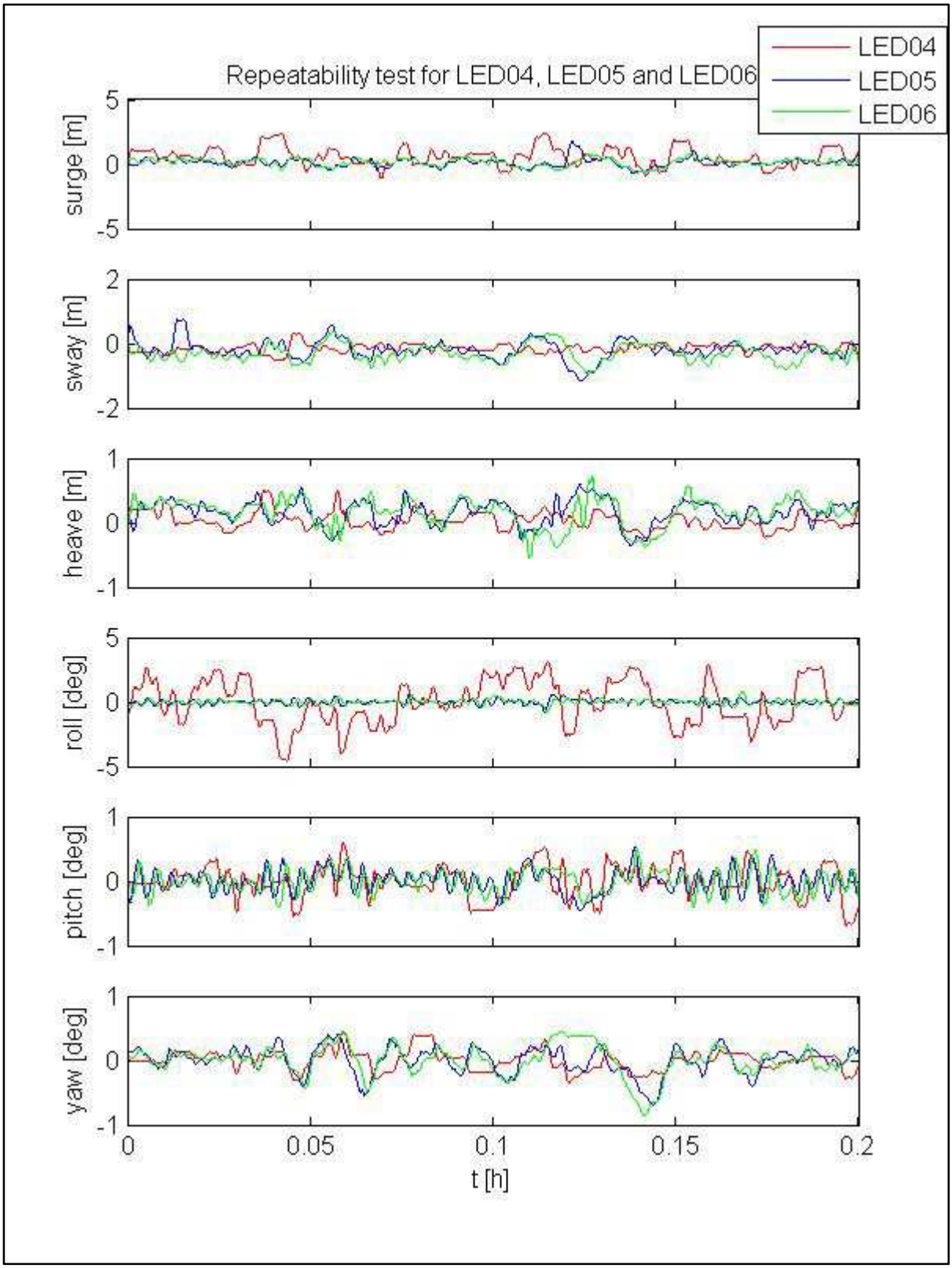

Figure 8.5: Repeatability plots for tests LED04, LED05 and LED06

8.2.3 Wave condition (H_{m0} of 1.5 m and T_P of 12 seconds prototype)

Using test LED08, the accuracy difference between the 2D LED tracking system and Keoship method are quantified in Table 8.6, and graphically shown in Figure 8.6. The accuracy difference between the three repeatability Tests LED07, LED08 and LED09 are quantified in Table 8.7, and graphically shown in Figure 8.7.

Table 8.6: Accuracy difference between 2D LED tracking and Keoship system, using Test LED08

Motion	R_{rms} (model scale)
Surge [mm]	9.96
Sway [mm]	6.84
Heave [mm]	6.32
Roll [°]	0.19
Pitch [°]	0.09
Yaw [°]	0.04

Table 8.7: Accuracy difference between 2D LED tracking tests for LED07, LED08 and LED09

Motion	R_{rms} (model scale) for LED07 and LED08	R_{rms} (model scale) for LED07 and LED09	R_{rms} (model scale) for LED08 and LED09
Surge [mm]	7.70	21.75	19.25
Sway [mm]	4.80	12.55	12.30
Heave [mm]	5.25	9.40	9.85
Roll [°]	0.11	0.20	0.18
Pitch [°]	0.07	0.10	0.09
Yaw [°]	0.06	0.06	0.07

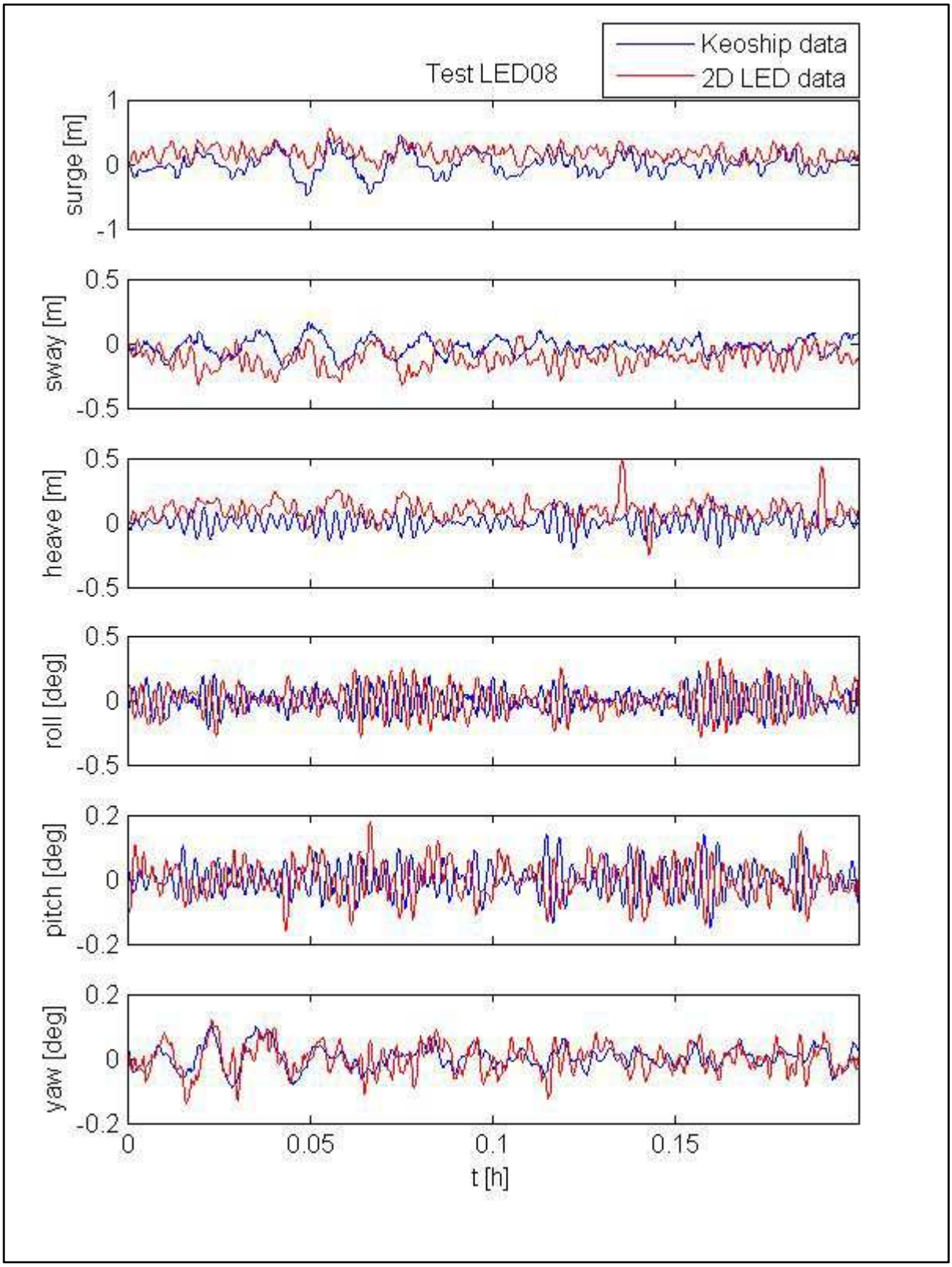

Figure 8.6: Data comparison for test LED08

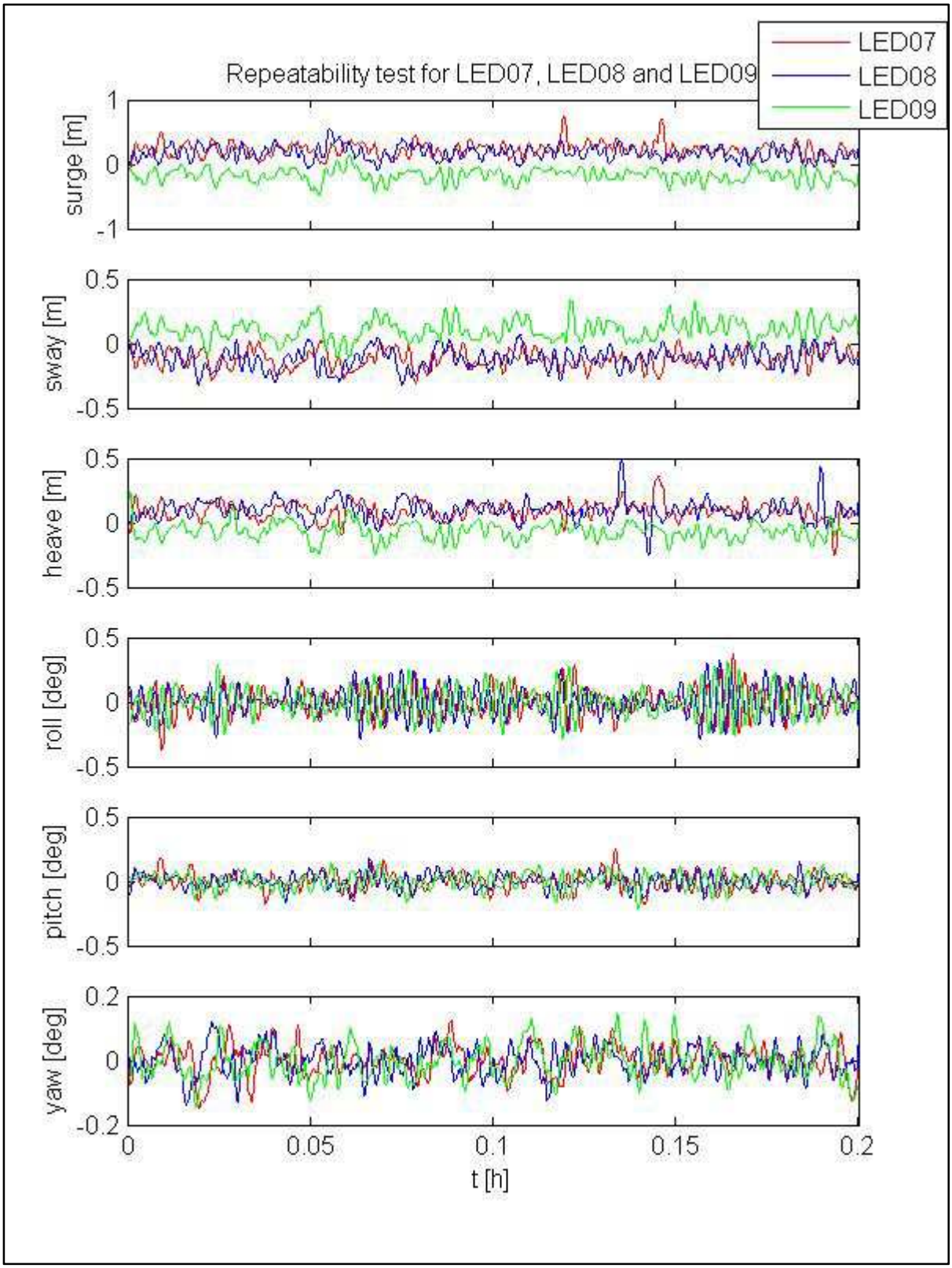

Figure 8.7: Repeatability plots for tests LED07, LED08 and LED09

8.2.4 Wave condition (H_{m0} of 3.0 m and T_P of 12 seconds prototype)

Using test LED10, the accuracy difference between the 2D LED tracking system and Keoship method are quantified in Table 8.8, and graphically shown in Figure 8.8. The accuracy difference between the two repeatability Tests LED10 LED12 are quantified in Table 8.9, and graphically shown in Figure 8.9.

Due to severe movement Test LED11 had no usable data.

Table 8.8: Accuracy difference between 2D LED tracking and Keoship system, using Test LED10

Motion	R_{rms} (model scale)
Surge [mm]	10.01
Sway [mm]	10.14
Heave [mm]	7.86
Roll [°]	0.30
Pitch [°]	0.15
Yaw [°]	0.17

Table 8.9: Accuracy difference between 2D LED tracking tests for LED010 and LED12

Motion	R_{rms} (model scale) for LED10 and LED12
Surge [mm]	18.60
Sway [mm]	22.25
Heave [mm]	6.80
Roll [°]	0.29
Pitch [°]	0.08
Yaw [°]	0.11

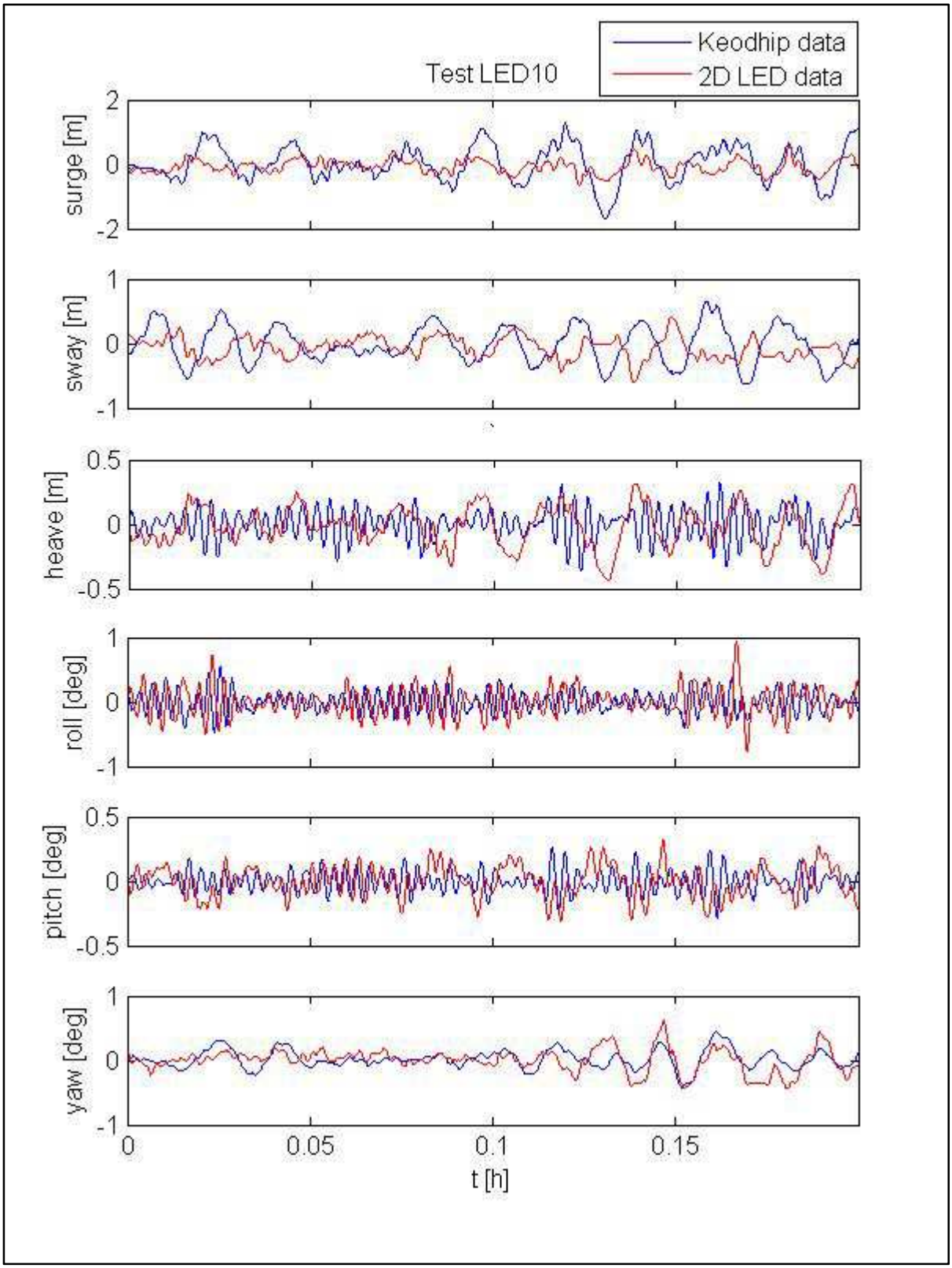

Figure 8.8: Data comparison for test LED10

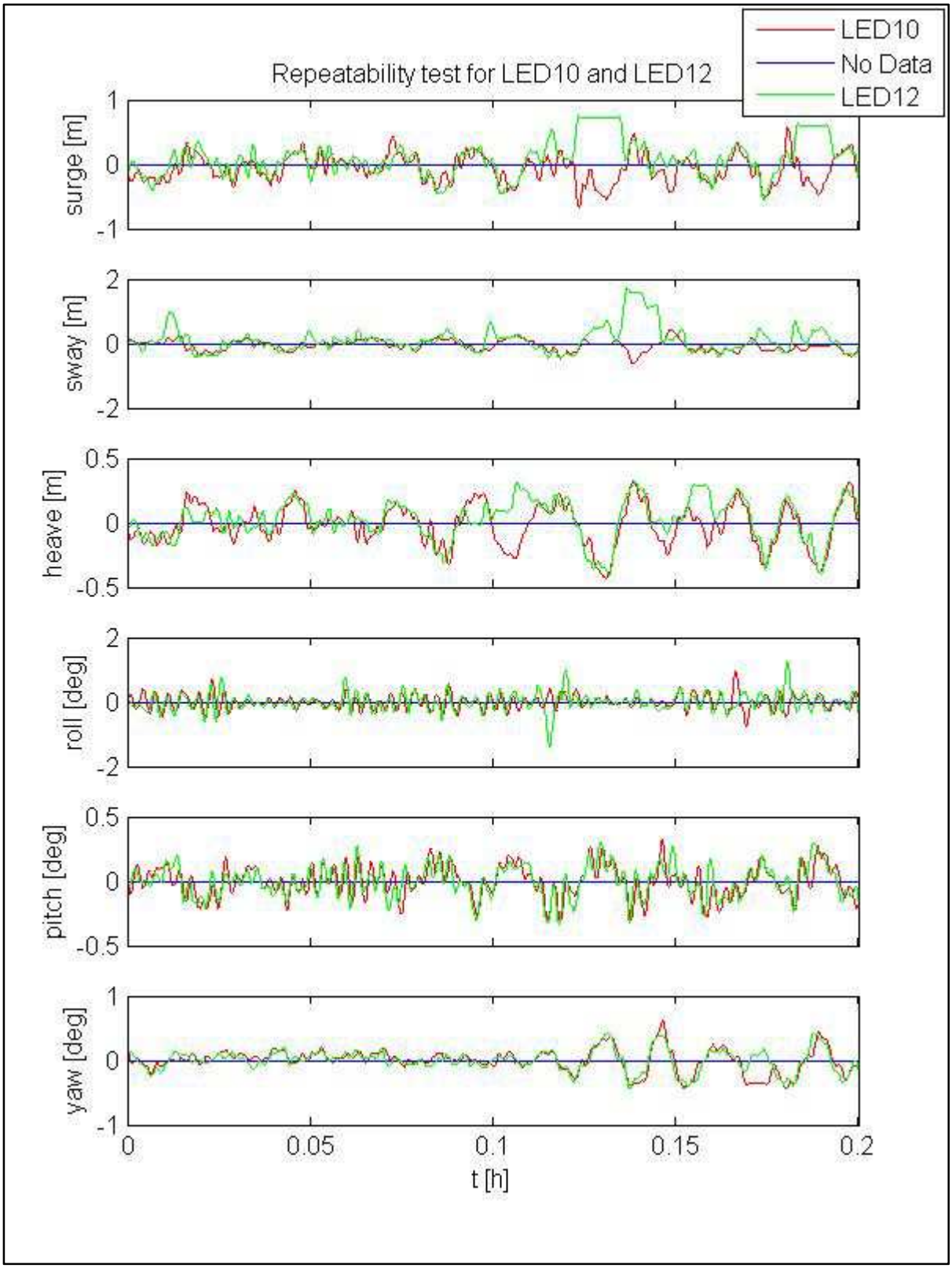

Figure 8.9: Repeatability plots for tests LED10 and LED12

8.3 Comparison between 3D Object Tracking and Keoship method

Like the 2D LED measurement data, the 3D object data exhibited unwanted transient points that needed to be discarded. All data points falling outside the maximum expected motions between image frames were removed. With the expected translation being less than 10 mm and rotation less than 1.0° between images (with a frame rate of 30 fps). The data were also filtered with a low pass filter in an effort to get rid of all the high frequency noise.

To accurately quantify the differences between the data sets, the root-mean-square error values are calculated before this gain adjustment and therefore appear different to what is shown in the comparison plots.

8.3.1 Wave condition (H_{m0} of 1.5 m and T_P of 16 seconds prototype)

Using test OBJ02, the accuracy difference between the 3D Object tracking system and Keoship method are quantified in Table 8.10, and graphically shown in Figure 8.10. The accuracy difference between the three repeatability Tests OBJ01, OBJ02 and OBJ03 are quantified in Table 8.11, and graphically shown in Figure 8.11.

Table 8.10: Accuracy difference between 3D Object tracking and Keoship system, using Test OBJ02

Motion	R_{rms} (model scale)
Surge [mm]	5.59
Sway [mm]	5.76
Heave [mm]	4.75
Roll [°]	0.18
Pitch [°]	0.14
Yaw [°]	0.13

Table 8.11: Accuracy difference between 3D Object tracking tests for OBJ01, OBJ02 and OBJ03

Motion	R_{rms} (model scale) for OBJ01 and OBJ02	R_{rms} (model scale) for OBJ01 and OBJ03	R_{rms} (model scale) for OBJ02 and OBJ03
Surge [mm]	16.00	15.40	4.80
Sway [mm]	12.10	9.75	6.20
Heave [mm]	8.50	8.80	3.50
Roll [°]	0.24	0.21	0.14
Pitch [°]	0.18	0.27	0.22
Yaw [°]	0.24	0.23	0.18

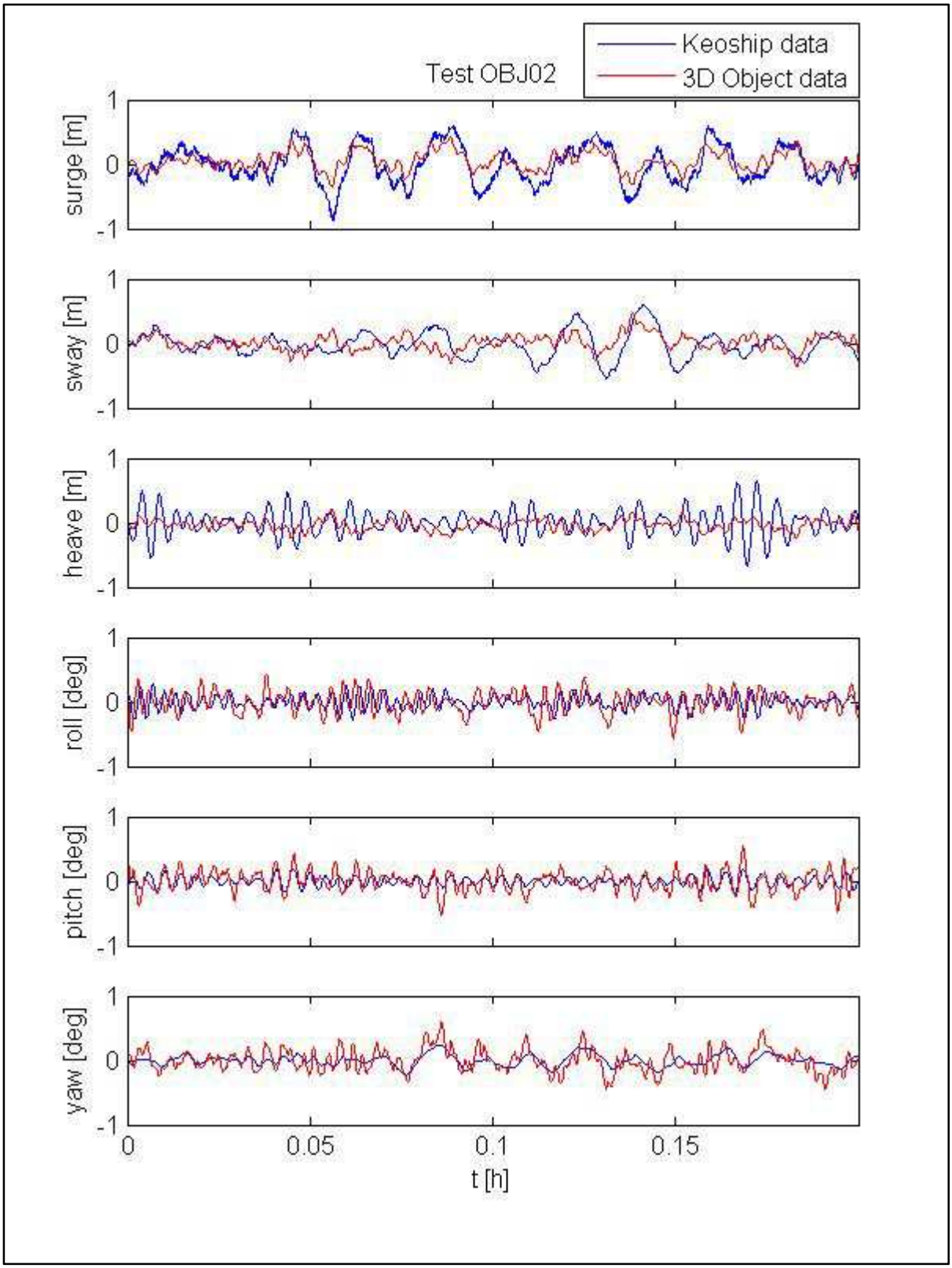

Figure 8.10: Data comparison for test OBJ02

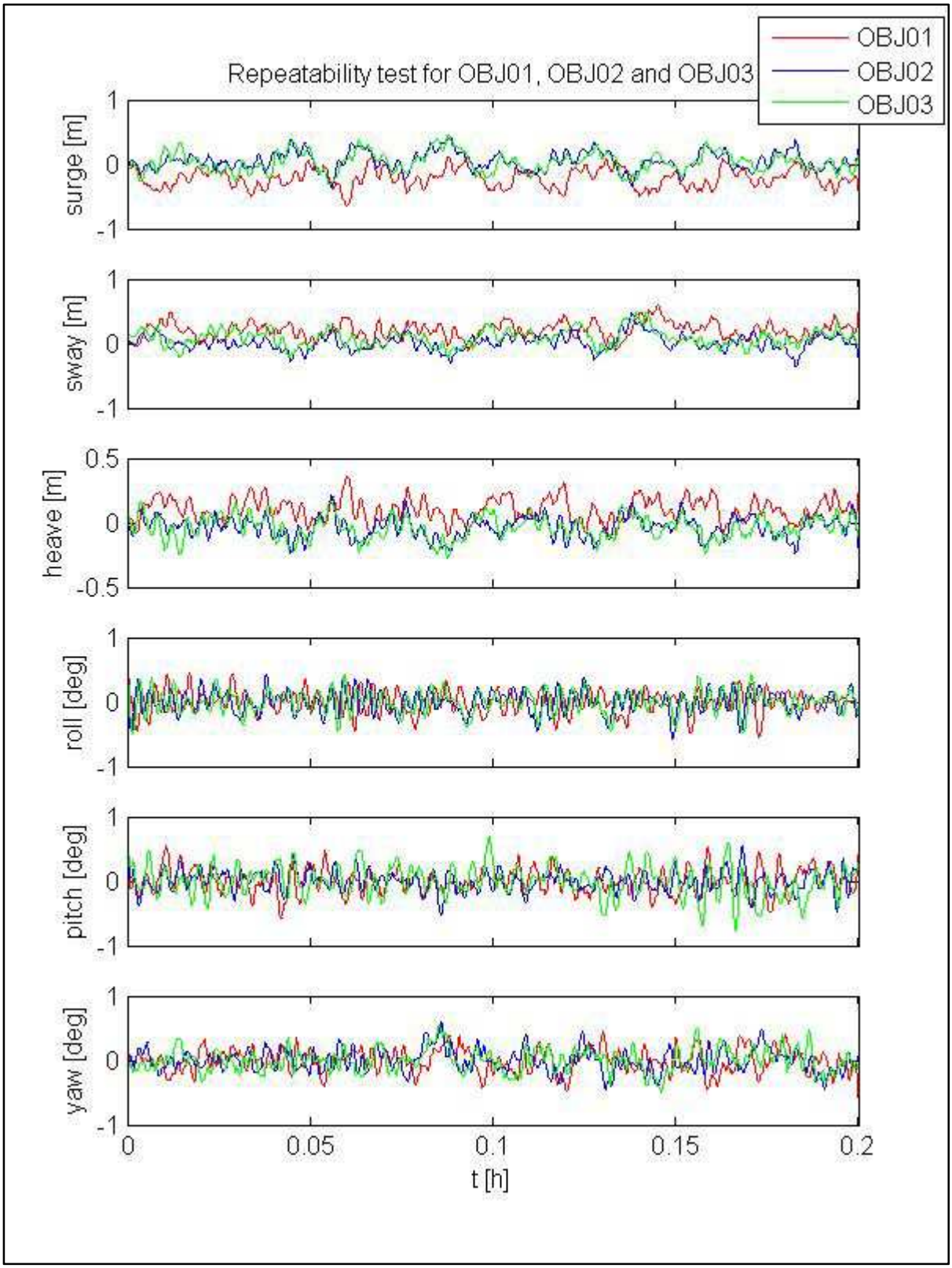

Figure 8.11: Repeatability plots for tests OBJ01, OBJ02 and OBJ03

8.3.2 Wave condition (H_{m0} of 3.0 m and T_P of 16 seconds prototype)

Using test OBJ04, the accuracy difference between the 2D LED tracking system and Keoship method are quantified in Table 8.10, and graphically shown in Figure 8.12. The accuracy difference between the three repeatability Tests OBJ04, OBJ05 and OBJ06 are quantified in Table 8.12, and graphically shown in Figure 8.13.

Table 8.12: Accuracy difference between 3D Object tracking and Keoship system, using Test OBJ04

Motion	R_{rms} (model scale)
Surge [mm]	14.27
Sway [mm]	16.75
Heave [mm]	11.43
Roll [°]	0.20
Pitch [°]	0.20
Yaw [°]	0.17

Table 8.13: Accuracy difference between 3D Object tracking tests for OBJ04, OBJ05 and OBJ06

Motion	R_{rms} (model scale) for OBJ04 and OBJ05	R_{rms} (model scale) for OBJ04 and OBJ06	R_{rms} (model scale) for OBJ05 and OBJ06
Surge [mm]	37.10	11.30	31.55
Sway [mm]	24.10	11.60	24.80
Heave [mm]	21.45	7.25	17.50
Roll [°]	0.31	0.35	0.32
Pitch [°]	0.33	0.35	0.24
Yaw [°]	0.39	0.25	0.34

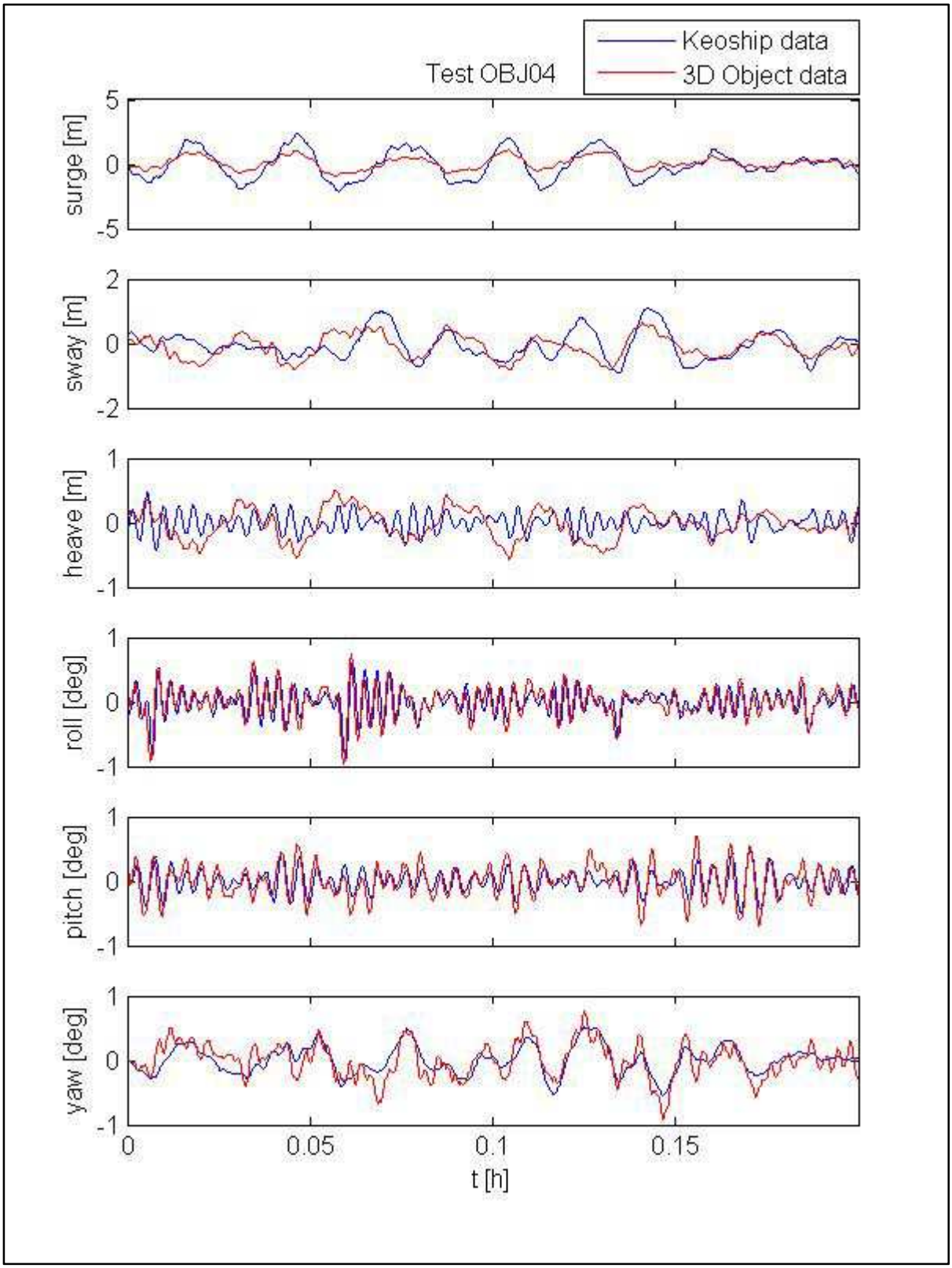

Figure 8.12: Data comparison for test OBJ04

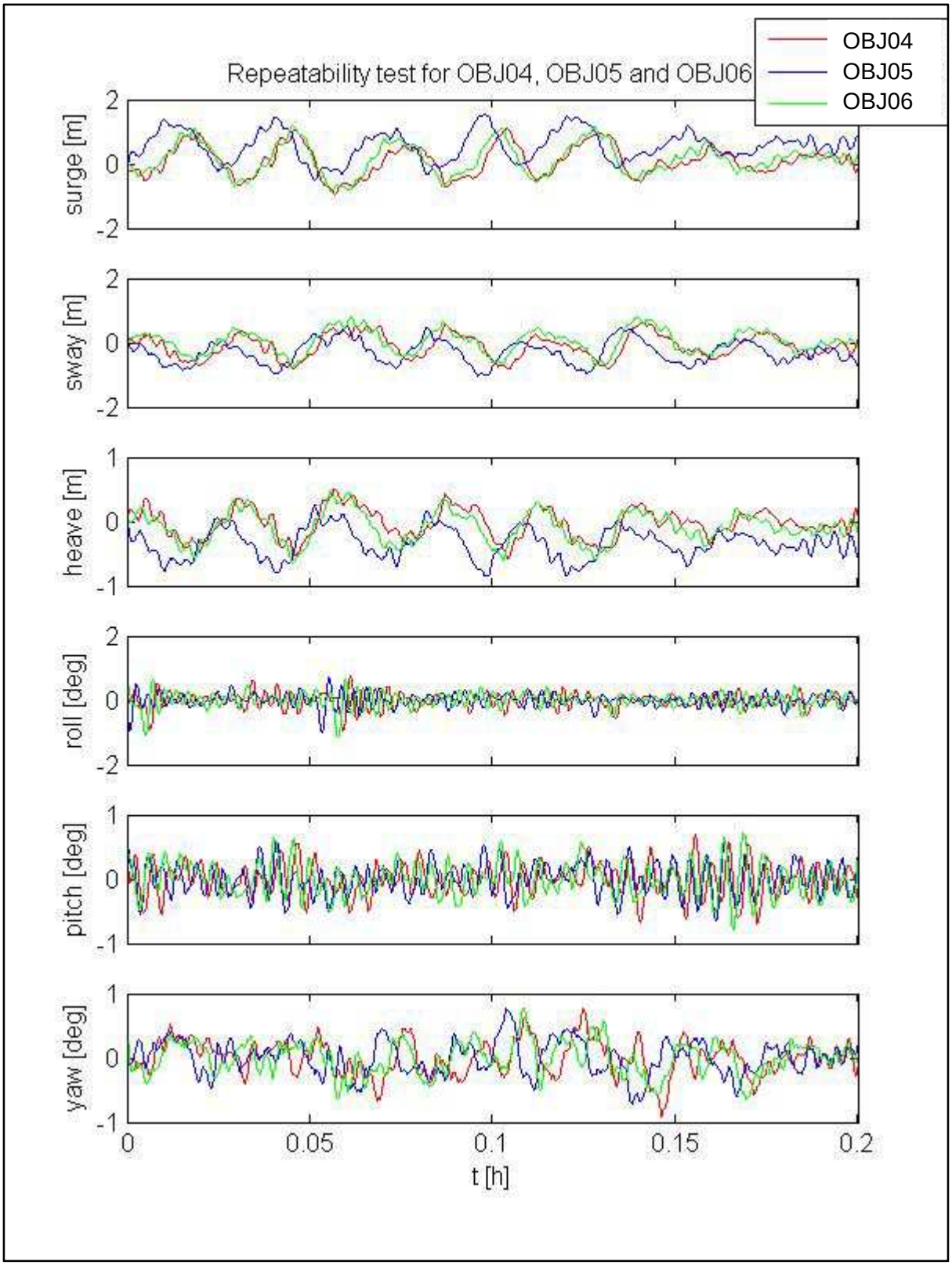

Figure 8.13: Repeatability plots for tests OBJ04, OBJ05 and OBJ06

8.3.3 Wave condition (H_{m0} of 1.5 m and T_P of 12 seconds prototype)

Using test OBJ07, the accuracy difference between the 3D Object tracking system and Keoship method are quantified in Table 8.14, and graphically shown in Figure 8.14. The accuracy difference between the three repeatability Tests OBJ07, OBJ08 and OBJ09 are quantified in Table 8.15, and graphically shown in Figure 8.15.

Table 8.14: Accuracy difference between 3D Object tracking and Keoship system, using Test OBJ07

Motion	R_{rms} (model scale)
Surge [mm]	5.37
Sway [mm]	6.58
Heave [mm]	4.37
Roll [°]	0.23
Pitch [°]	0.15
Yaw [°]	0.13

Table 8.15: Accuracy difference between 3D Object tracking tests for OBJ07, OBJ08 and OBJ09

Motion	R_{rms} (model scale) for OBJ07 and OBJ08	R_{rms} (model scale) for OBJ07 and OBJ09	R_{rms} (model scale) for OBJ08 and OBJ09
Surge [mm]	13.00	6.50	14.15
Sway [mm]	16.55	5.85	17.00
Heave [mm]	11.05	4.85	12.65
Roll [°]	0.22	0.22	0.00
Pitch [°]	0.18	0.18	0.01
Yaw [°]	0.17	0.17	0.01

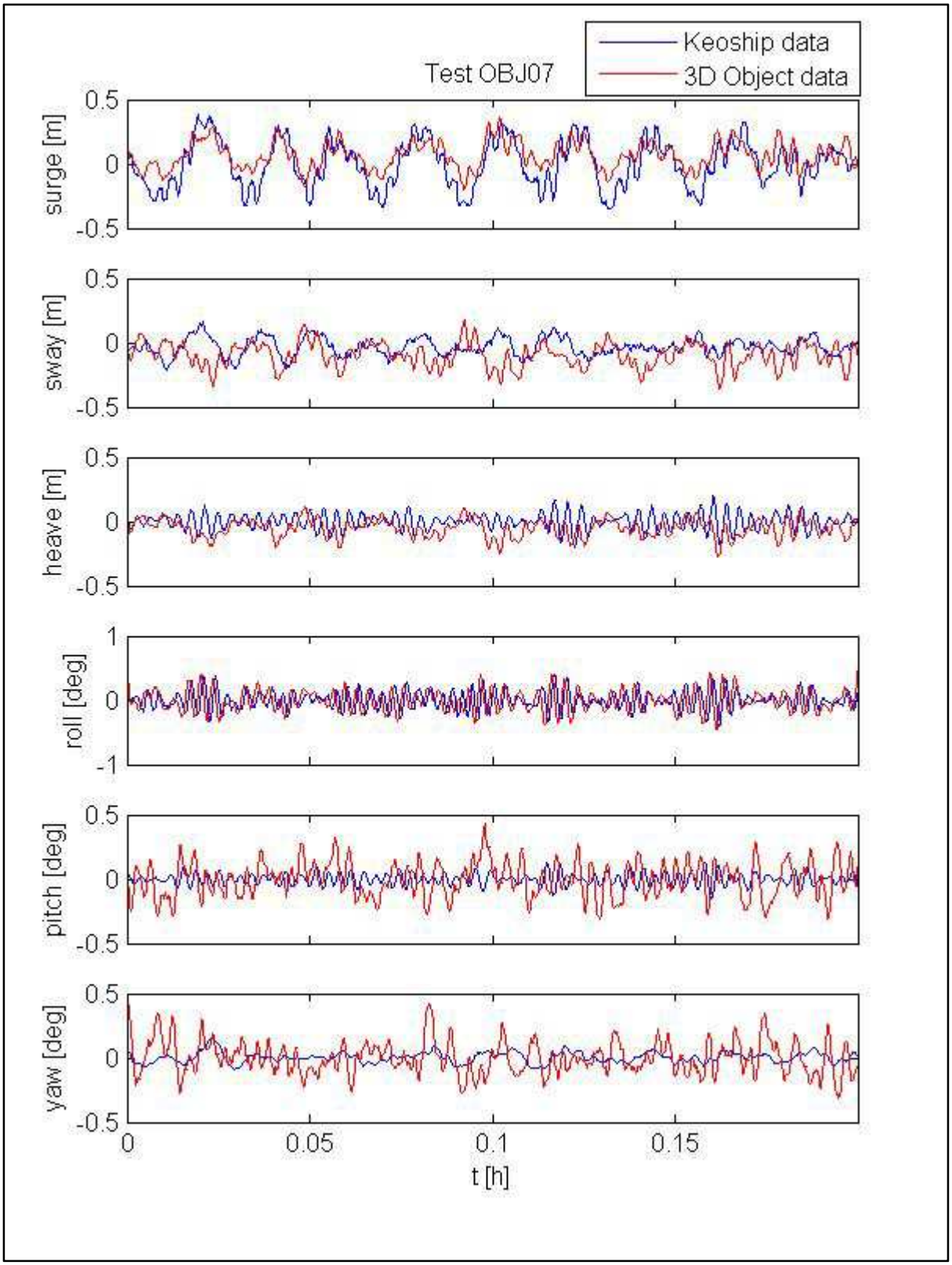

Figure 8.14: Data comparison for test OBJ07

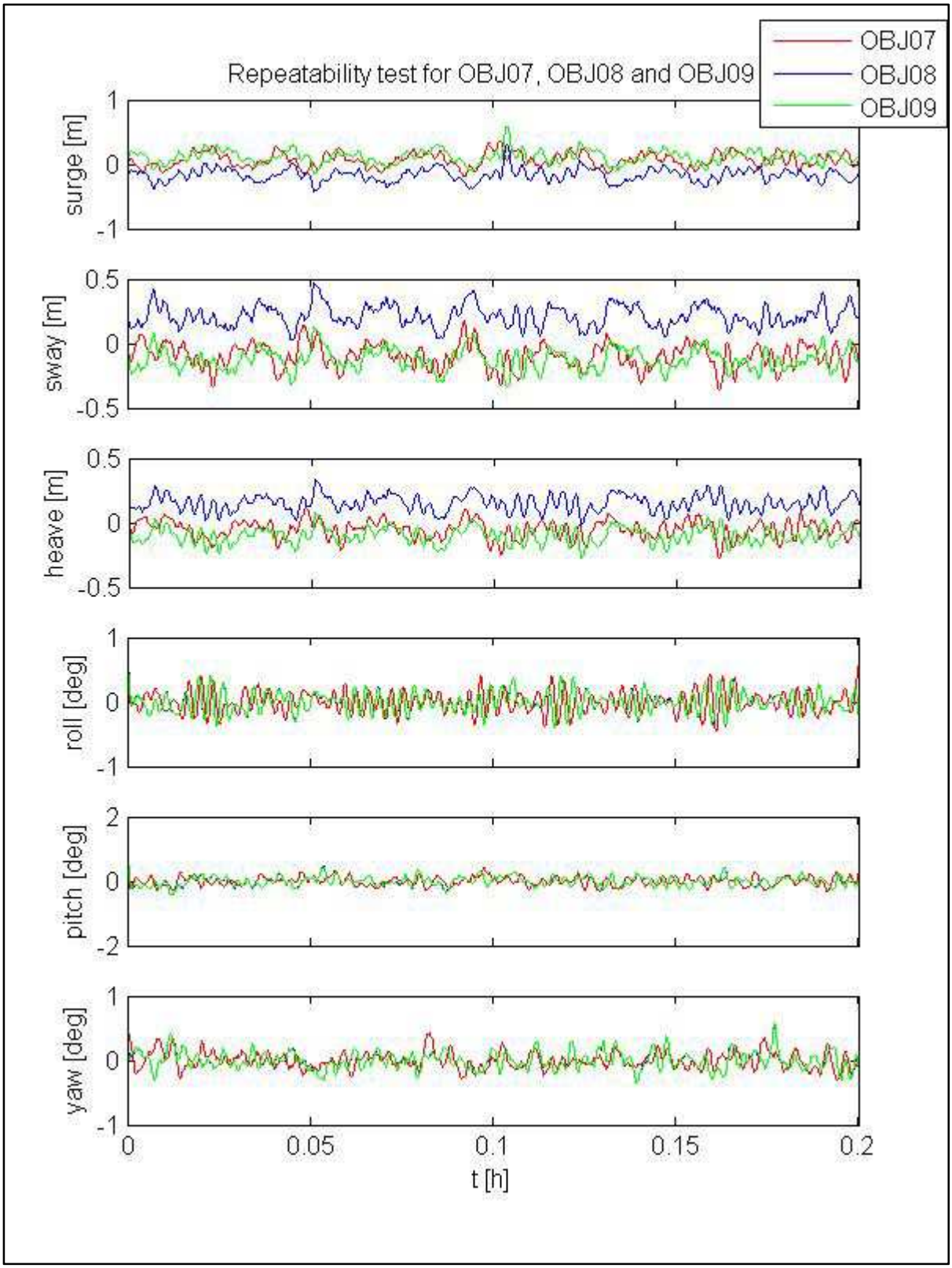

Figure 8.15: Repeatability plots for tests OBJ07, OBJ08 and OBJ09

8.3.4 Wave condition (H_{m0} of 3.0 m and T_P of 12 seconds prototype)

Using test OBJ11, the accuracy difference between the 2D LED tracking system and Keoship method are quantified in Table 8.16, and graphically shown in Figure 8.16. The accuracy difference between the three repeatability Tests OBJ10, OBJ11 and OBJ12 are quantified in Table 8.17, and graphically shown in Figure 8.17.

Table 8.16: Accuracy difference between 3D Object tracking and Keoship system, using Test OBJ11

Motion	R_{rms} (model scale)
Surge [mm]	9.66
Sway [mm]	10.06
Heave [mm]	5.78
Roll [°]	0.29
Pitch [°]	0.16
Yaw [°]	0.18

Table 8.17: Accuracy difference between 3D Object tracking tests for OBJ10, OBJ11 and OBJ12

Motion	R_{rms} (model scale) for OBJ04 and OBJ05	R_{rms} (model scale) for OBJ04 and OBJ06	R_{rms} (model scale) for OBJ05 and OBJ06
Surge [mm]	15.05	18.79	13.46
Sway [mm]	8.90	11.12	9.89
Heave [mm]	9.57	8.60	7.33
Roll [°]	0.40	0.37	0.29
Pitch [°]	0.24	0.24	0.19
Yaw [°]	0.34	0.32	0.24

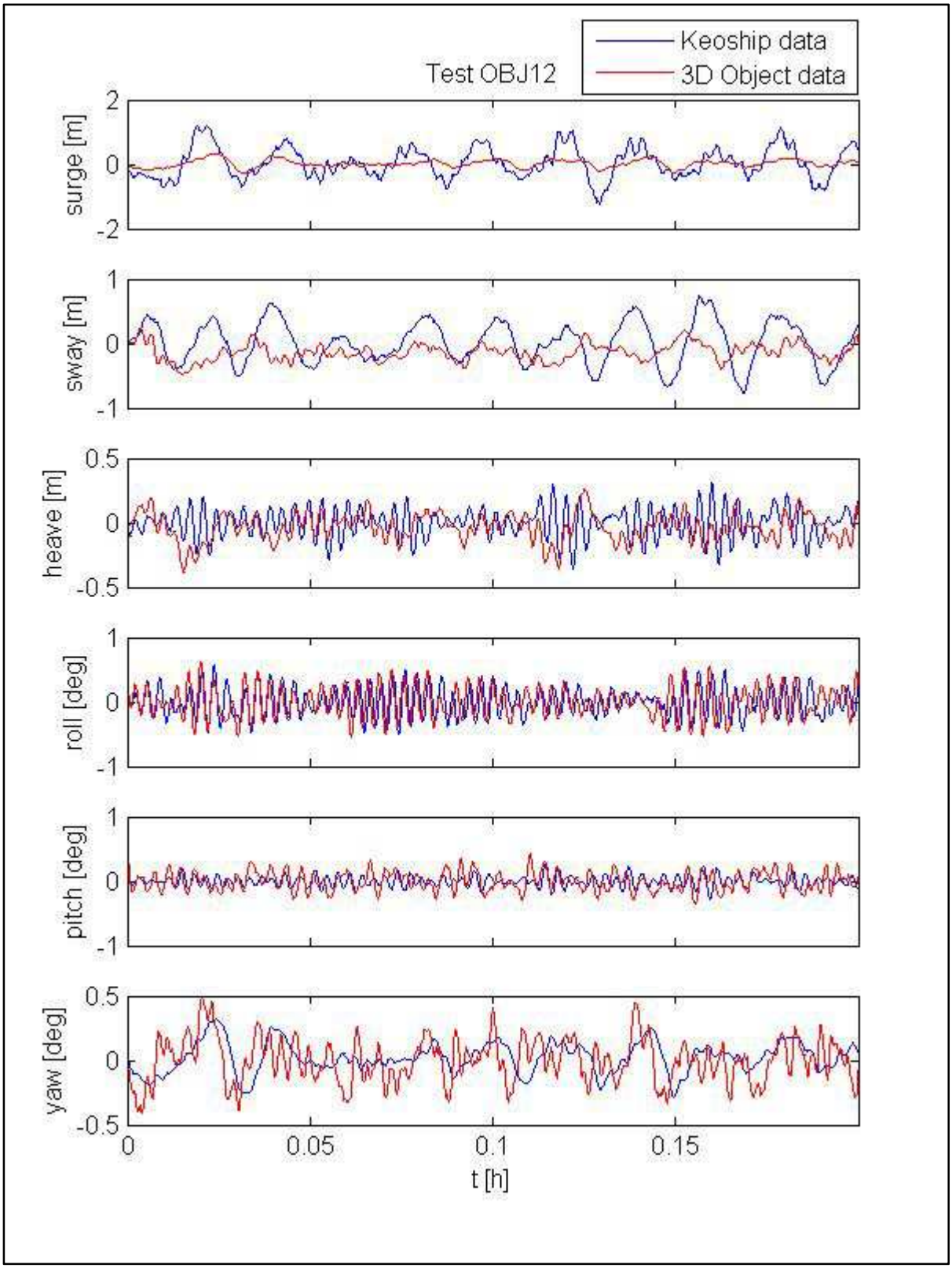

Figure 8.16: Data comparison for test OBJ11

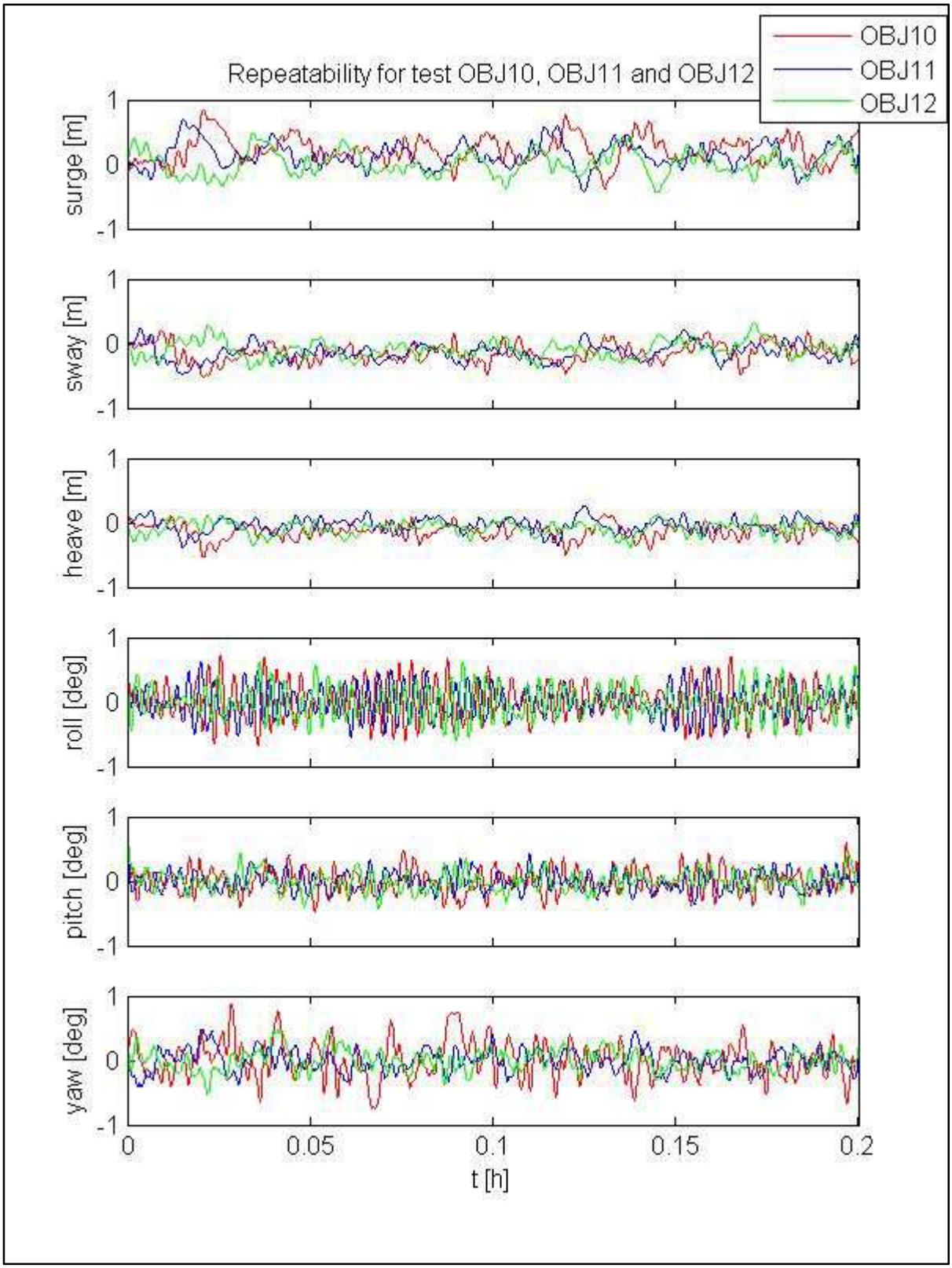

Figure 8.17: Repeatability plots for tests OBJ10, OBJ11 and OBJ12

9 DISCUSSION

This section discusses the results obtained from the static verification and accuracy tests as well as for the physical model testing.

9.1 Verification and Accuracy Testing

For all practical purposes, the 2D LED tracking system is ignored in this discussion due to the error in depth estimation, resulting in transient data.

As stated in section 2.2.4, the maximum critical significant surge and sway amplitude for an (un)loading efficiency of 95% is 0.4 m. The conservative rule of thumb, assuming mooring lines can break at surge or sway motions equal to 1.2 m or larger, is also used. The accuracy tests for translation were conducted for displacements ranging from 10 mm to 1500 mm, thus covering motion for normal working conditions as well as conditions which can cause mooring lines to break.

The maximum significant motion amplitude given in PIANC (2012), is provided to one decimal place. It can be assumed by using this, when analysis of moored container vessel motions is required, an accuracy criterion of 0.1 m is acceptable for surge and sway motions.

Figure 9.1 incorporates the surge and sway results obtained from the accuracy tests as well as the accuracy criterion to help establish the usability of the tracking system developed by the author, at its current development stage.

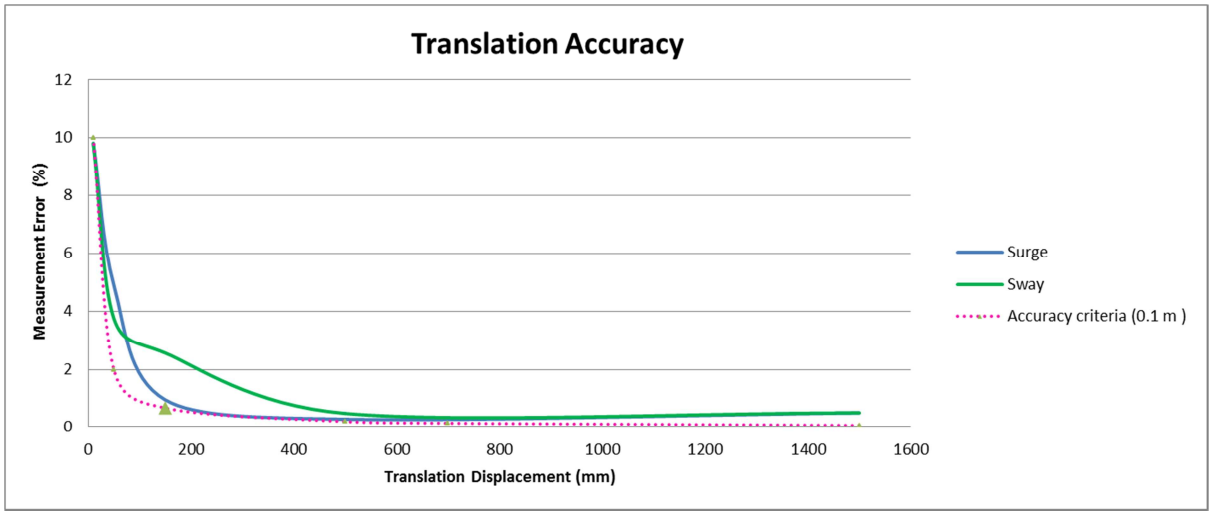

Figure 9.1: Accuracy criteria incorporated with surge and sway results obtained from the accuracy tests

The graph indicates that when using the accuracy criterion of 0.1 m the tracking system would be accurate enough to be used when small measurements are of interest (i.e. a physical model study). For a model study however, scaling also has to be considered which for a 1:100 scale model, requires a model accuracy of at least 1 mm to fall within the prototype accuracy criterion of 0.1 m.

It is thus possible to state that at the current development stage, neither the 2D LED tracking system or 3D Object tracking system are accurate enough for use in a physical model study.

The graph also indicates however, that for prototype motions exceeding 0.6 m (i.e. storm events) the 3D Object tracking system would have an accuracy close to the maximum allowable criterion of 0.1 m. This makes the system usable at its current development stage when rapid deployment during a storm event is needed in prototype conditions.

In situations where funds and time are available for the deployment of a D-GPS system, it would currently be a better option to utilise a D-GPS system, seeing that it has an accuracy of about 2 cm (CSIR et al. 2013).

9.2 Physical Model Results for 2D LED Tracking

9.2.1 Translation measurements

For surge and sway, when comparing the 2D LED tracking data to the Keoship data, it was found that with an increase in wave height, the root-mean-square error difference becomes larger. The error difference between the 1.5 m and 3.0 m wave condition was generally found to be greater for the 16 s wave period tests than for the 12 s period tests.

The comparison plots show that even for the Keoship measurements, vessel motions were physically smaller for the shorter period waves than for the larger period waves. The smaller accuracy difference between the 1.5 m and 3.0 m wave condition for the 12 s wave period is most likely due to the motions being physically smaller for the same input wave height. It is therefore not possible to quantify the accuracy change across different wave periods for a change in wave height.

For the 16 s wave period tests, the increase in error difference is from a maximum R_{rms} value of 11.60 mm to 20.38 mm. An increase in error difference is also evident for the 12 second wave period tests, changing from a maximum R_{rms} value of 9.96 mm to 10.14 mm.

For heave, although the root-mean square differences calculated follow the same trend as for the surge and sway, it can unfortunately be seen from the comparison plots that the tracking algorithm loses track of the smaller frequency heave motions, and rather provides a smoothed estimate. From this smoothed representation, only the maximums can be used.

9.2.2 Rotation measurements

For roll, pitch and yaw the 2D LED tracking data compared well to the Keoship data. From the comparison plots and the root-mean-square error differences it is possible to see that the rotations are more accurately measured when the rotations are small. A maximum R_{rms} value of 0.19° for motions smaller than 0.5° and a maximum R_{rms} value of 0.31° for motions above 0.5° were calculated. On average, pitch and yaw motions were found to have a R_{rms} value of 0.20° smaller than the R_{rms} values obtained for roll, making the pitch and yaw estimates more accurate.

9.2.3 Measurement resolution

The resolution for the measurement system is estimated by evaluating the maximum R_{rms} values between repeatability tests. For each repeatability series, the average R_{rms} value calculated for each of the 6DOF motions is taken as a conservative estimate of the resolution for measuring that particular motion. These estimates are presented in Table 9.1.

Table 9.1: Estimated resolution accuracy for 2D LED tracking physical model tests.

Motion	Resolution Estimates (in model scale)			
	H_{m0} =1.5 m T_p = 16s	H_{m0} =3.0 m T_p = 12s	H_{m0} =1.5 m T_p = 12s	H_{m0} =3.0 m T_p = 12s
Surge [mm]	9.01	33.32	16.23	18.6
Sway [mm]	5.88	16.31	9.88	22.25
Heave [mm]	6.36	11.73	8.17	6.80
Roll [°]	0.15	1.21	0.16	0.29
Pitch [°]	0.16	0.23	0.09	0.08
Yaw [°]	0.19	0.21	0.06	0.11

9.3 Physical Model Results for 3D Object Tracking

9.3.1 Translation measurements

As for the 2D LED tracking data, it was found that for surge and sway an increase in wave height increases the root-mean-square error difference for the individual motions.

For the 16 second wave period tests, the increase in error difference is from a maximum R_{rms} value of 5.76 mm to 16.76 mm. An increase in error difference is also evident for the 12 second wave period tests, changing from a maximum R_{rms} value of 6.58 mm to 10.06 mm.

For heave, as for the 2D LED tracking data, the comparison plots between the 3D Object tracking data and Keoship data show that the tracking algorithm loses track of the smaller frequency heave motions.

From the R_{rms} values and the comparison plots, it is evident that the translation results obtained from the 3D Object tracking tests compare better to the Keoship motions than the 2D LED tracking results. On average the error difference is 2.40 mm more accurate for surge, 2.45 mm for sway and 2.27 mm for heave.

9.3.2 Rotation measurements

For roll, pitch and yaw the 3D Object tracking data compared extremely well to the Keoship data. From the comparison plots and the root-mean-square error differences, it is possible to see that the rotations are more accurately measured when the rotations are small. A maximum R_{rms} value of 0.23° for motions smaller than 0.5° and a maximum R_{rms} value of 0.29° for motions above 0.5° were calculated. On average, pitch and yaw motions were found to have a R_{rms} value of 0.07° smaller than the R_{rms} values obtained for roll, making the pitch and yaw estimates more accurate.

Although the rotation error differences for both tracking systems are similar, visual verification of the comparison plots indicate that the 3D object tracking system better compare to the corresponding Keoship data.

9.3.3 Measurement resolution

Using the same evaluating criteria as for the 2D LED tracking results, the measurement resolution is estimated for the 3D object tracking system and is presented in table Table 9.2.

Table 9.2: Estimated resolution accuracy for 3D Object tracking physical model tests.

Motion	Resolution Estimates (in model scale)			
	H_{m0} =1.5 m T_p = 16s	H_{m0} =3.0 m T_p = 12s	H_{m0} =1.5 m T_p = 12s	H_{m0} =3.0 m T_p = 12s
Surge [mm]	12.07	26.65	9.27	15.77
Sway [mm]	9.35	20.17	13.13	9.97
Heave [mm]	6.93	15.4	9.52	10.20
Roll [°]	0.20	0.33	0.17	0.35
Pitch [°]	0.22	0.31	0.12	0.22
Yaw [°]	0.22	0.32	0.12	0.45

9.4 Comparison Plots for Physical Model Results

The amplitude gain difference for the translation results, is still an unresolved issue. This requires further work to establish why there is a gain difference. This phenomenon is not present with any of the static verification or accuracy tests. The physical model tests, which are dynamic in nature, do however include this phenomenon.

The verification and tracking test results show good data for the current development stage. These verified that the 3D tracking system works and can be developed further. It will be important to determine the reason behind this difference before continuing to the next stage of development, which includes refinement of the image processing and tracking algorithms to achieve better accuracies over a range of displacements.

10 CONCLUSION

From the verification and accuracy results, it is evident to conclude that the 2D LED tracking system should not be further pursued and is not a viable solution for use in a physical model or in the field.

The accuracy tests done on the 3D Object tracking system indicate that at the current development stage, the accuracy is unfortunately not good enough to use in a physical model study.

For the physical model tests, the accuracy of the 3D Object tracking system was in the order of 5.76 - 16.76 mm for translations and 0.23-0.29° for rotation which is much greater than the allowable accuracy criterion of 1 mm when considering a model scale of 1:100.

At the current development stage, the Keoship system from the CSIR is still the most accurate for measurements involving physical model tests. The accuracy of the system is 0.1 mm for translations and 0.04° for roll and 0.01° for pitch and yaw (CSIR et al. 2008).

The Keoship system is intrusive and is therefore bound to the physical modelling environment. The verification results indicate that for prototype motions exceeding 0.6 m, the 3D object tracking system, at the current development stage, would have acceptable accuracy close to the allowable accuracy criterion of 0.1 m. When rapid deployment is needed for measurements during a storm event, opting for the non-intrusive method will be an acceptable compromise even though the accuracy of an intrusive D-GPS system is in the order of 2 cm when ultimately deployed.

At the current development stage, the system can however only be used as a post processing tool.

11 FUTURE DEVELOPMENTS

The system should be changed to allow for real time processing. This can be done by running the image point extraction algorithm on the live feed and only saving the points of interest to a text file.

The motions could then also be processed and displayed real time by making a few computational improvements and only using one programming platform to do the image processing and 6DOF calculations (currently Matlab and C sharp are used).

The POSIT algorithm that was used, can be compared to other pose estimation algorithms in order to determine if there is another algorithm that can estimate pose more accurately. Refinement on the Matlab algorithm which locates the image coordinates of the object, could also be looked at to enable sub-pixel accuracy.

Focal length which is one of the main input parameters to the POSIT algorithm, was determined using a simple pinhole camera model. It is suggested that a more sophisticated model should be used to determine focal length.

Lens distortion calculated during the camera calibration process is currently not included in the pose estimation process and should be incorporated into the pose estimation algorithm.

In order to avoid lighting effects, which would be a concern for prototype outdoor use, it is suggested that an infrared camera receiver should be used to record image displacements.

It is envisioned that auxiliary vision systems can play a useful role in monitoring maritime operations in the port area, especially while vessels are berthed. With the aid of high resolution camera systems, it will be possible to automatically provide additional visual information as well as give early warning to vessels that experience motions beyond a certain allowable criteria.

From an algorithm development point of view there are a lot of possibilities. A port could obtain the outer shell dimensions of those ships that frequently visit the port from the respective ship owners (i.e. the outer dimensions of the bridge structure). A database with the 3D mesh information for frequently visiting ships will then be available. The algorithm which currently locates the corner points of a known object could be altered to first find the best possible match to a 3D mesh structure. The relevant image points to the best possible object match on the image could then be used with the current pose estimation algorithm.

A further possible use for the system could be to evaluate under keel clearance for ships entering and exiting a port. The tracking of vessel motions for a ship would be the same as mentioned above. A port operator would then input current tidal conditions and vessel draft information into a GUI. The GUI will then use the real time motions recorded with the parameters provided as input by the port operator and estimate real time under keel clearance.

12 REFERENCES

Bebis, G., 2004. Computer Vision, Class notes on camera parameters Course CS491E, Computer Science Department, University of Nevada, Reno in 2004. , (1), pp.1–9.

Benetazzo, A., 2011. Accurate measurement of six Degree of Freedom small-scale ship motion through analysis of one camera images. *Ocean Engineering*, 38(16), pp.1755–1762. Available at: http://linkinghub.elsevier.com/retrieve/pii/S0029801811001818 [Accessed August 26, 2014].

Briggs, M. & Melito, I., 2008. Barbers Point Harbor , Hawaii , Jetty Modification Study Coastal and Hydraulics Laboratory Barbers Point Harbor , Hawaii ,. , (November).

Corke, P.I., 2011. Homogeneous transform to roll-pitch-yaw angles [Computer program].Available at https://code.google.com/p/matlab-toolboxes-robotics-vision/source/browse/matlab/robot/tags/R9.4/tr2rpy.m?r=822 (Accessed 11 September 2013).

CSIR et al., 2008. *Khalifa Port and Industrial Zone, Physical Modelling of Harbour Agitation and Moored Ship Dynamics*, Stellenbosch.

CSIR, Kieviet, J. & Rossouw, M., 2013. *Ship Motion Measurement Exercise for the Port of Ngqura, Navigation and Mooring Safety Study*, Stellenbosch.

DigitalRune, Pose - Position and Orientation. Available at: http://www.digitalrune.com/Documentation/html/d995ee69-0650-4993-babd-1cdb1fd8fd7a.htm#SectionPoseReasons [Accessed August 26, 2014].

Hughes, S.A., 1993. *Physical models and laboratory techniques in coastal engineering*, Singapore: World Scientific.

ITTC, 2008. International Towing Tank Conference, September 14-20 2008. The Ocean Engineering Committee. Final Report and Recommendations to the 25th ITTC. In Proceedings of the 25th International Towing Tank Conference, Fukuoka, Japan.

ITTC, 1999. International Towing Tank Conference, September 5-11 1999. The Loads and Responses Committee. Final Report and Recommendations to the 22nd ITTC. In: Proceedings of the 22nd International Towing Tank Conference, Seoul and Shanghai.

Journée, J.M.J. & Pinkster, J.A., 2001. *SHIP HYDROMECHANICS, Part I: Introduction* Draft Edit., Mekelweg 2, 2628 CD, Delft, The Netherlands: Ship Hydromechanics Laboratory, Delft University of Technology. Available at: http://www.shipmotions.nl.

Kinsman, B., 1965. *Wind Waves: Their Generation and Propagation on the Ocean Surface*, New Jersey: Prentice-Hall.

Kirillov, A., 2011. Coplanar PoseEstimation application [Computer program].Available at http://www.aforgenet.com/articles/posit (Accessed 10 September 2013).

Liang, O., 2013. Build A Quadcopter From Scratch - Hardware Overview. *OscarLiang.net*. Available at: http://blog.oscarliang.net/build-a-quadcopter-beginners-tutorial-1/ [Accessed August 26, 2014].

Malheiros, P. et al., 2009. Robust and real-time motion capture of rigid bodies based on stereoscopic vision. In *3rd International Conference on Integrity, Reliability and Failure*. 3rd International Conference on Integrity, Reliability and Failure. Porto, Portugal. Available at: http://repositorio-aberto.up.pt/handle/10216/68540 [Accessed August 26, 2014].

MARIN, Motion monitoring. Available at: http://www.marin.nl/web/Research-Topics/Offshore-operations/Motion-monitoring.htm [Accessed August 26, 2014].

De Menthon, D., 2003. ClasicPOSIT [Computer program].Available at http://www.cfar.umd.edu/~daniel/Site_2/Code.html (Accessed 11 September 2013).

Moes, J. & Hough, G., 1999. Measurment of Moored Ship Motions Using Video Imaging Technology. In *12th International Harbour Congress*. Antwerpen: KVIV.

Van der Molen, W. et al., 2010. Innovative technologies to accurately model waves and moored ship motions. In *CSIR Third Biennial Conference 2010*. Pretoria, South Africa: CSIR. Available at: http://researchspace.csir.co.za/dspace/handle/10204/4252 [Accessed August 26, 2014].

Van der Molen, W. & CSIR, 2010. Ship Hydrodynamics, Microsoft PowerPoint lecture notes on 5th lecture in Module W03-2 of post graduate Coastal Engineerg studies at the University of Stellenbosch in 2011.

PIANC, 2012. *Criteria for the (Un)Loading of Container Vessels. Report no. 115-2012, s.l.: s.n.*,

Van Son, S.T.J., 2008. *System for monitoring moored ship motions in Port of Saldanha*, Stellenbosch, South Africa: CSIR.

Stuart, D.C.A., 2013. *Characterizing long wave agitation in the Port of Ngqura*. University of Stellenbosch.

USACE, 2006. Coastal Engineering Manual. Engineer Manual 1110-2-1100, U.S. Army Corps of Engineers, Washington, D.C. (in 6 volumes).

Yoon, Y. et al., 2005. A new approach to the use of edge extremities for model-based object tracking. In *2005 IEEE International Conference on Robotics and Automation*. Barcelona, Spain, pp. 1883–1889. Available at: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=1570386 [Accessed August 26, 2014].

Yoon, Y., Kosaka, A. & Kak, A.C., 2008. A New Kalman-Filter-Based Framework for Fast and Accurate Visual Tracking of Rigid Objects. *IEEE Transactions on Robotics*, 24(5), pp.1238–1251. Available at: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=4631506.

Yves Bouguet, J., Camera Calibration Toolbox for Matlab. *Last updated 2 December 2013*. Available at: http://www.vision.caltech.edu/bouguetj/calib_doc/htmls/example.html [Accessed September 20, 2014].

Appendix A Image Processing Script for the 2D LED Tracking

```
%These scripts are project specific and calculates the image coordinates for
%the 2D LED target to be tracked in the thesis work of Johan Kieviet
%
%Copyright © CSIR 2013, program by Johan Kieviet (JK)and Jatin Harribhai (JH)
%

fid = fopen('pixel points.txt','w');
beginn=15824;
endd=17982;
imageXC= 2304;
imageYC= 1728;
for k=beginn:endd                                   %nzFrames
    filename = sprintf('Pic%d.jpg',k);              %added by JK

    imagergb = imread(filename,'jpg');              %changed by JK

    image = rgb2gray(imagergb);

    imageb = imagergb(:,:,3);      %Image3 is only BLUE layer of grey image

    image3 = imcrop(imageb,[[484 469 856 439]]); %% Starboard side "Nikon AW100"
    %%--Converting RGB image to BW image NB! Specifies intensities to keep
    %%---------------------------------------------------------------------
    bwimage2 = im2bw(image3, 0.8);

    %%--Eroding and Dilating of BW image sufficient times
    %%---------------------------------------------------------------------
    se1 = strel('disk',1);
    erodeBW = imerode(bwimage2,se1);

    se2 = strel('line',5,135);
    dilateBW = imdilate(erodeBW,se2);

    se3 = strel('line',5,45);
    dilateBW = imdilate(dilateBW,se3);

    se4 = strel('disk',2);
    dilateBW = imdilate(dilateBW,se4);

    imdil = 255 * dilateBW;

    corner_peaks = imregionalmax(imdil);
    corner_idx = find(corner_peaks == true);
    [r, g, b] = deal(image3);
    r(corner_idx) = 0;
    g(corner_idx) = 0;
    b(corner_idx) = 255;
    RGB = cat(3,r,g,b);

for k=beginn:(beginn+2)
    filename = sprintf('imageOverlay%d.jpg',k);      %added by JK
    imwrite(RGB,filename);                           %added by JK
end

for k=(endd-2):endd
    filename = sprintf('imageOverlay%d.jpg',k);      %added by JK
    imwrite(RGB,filename);                           %added by JK
end
    [row,col] = size(imdil);
    low_row = row;
    high_row = 1;
    low_col = col;
    high_col = 1;

    for i = 1:row
        for j = 1:col
            if (corner_peaks(i,j) == 1)
                if (i < low_row)
                    low_row = i;
                    point2 = [i j];
                else if ( i > high_row)
                        high_row = i;
                        point4 = [i j];
                    end
                end
```

```
                if (j < low_col)
                    low_col = j;
                    point1 = [i j];
                else if (j > high_col)
                        high_col = j;
                        point3 = [i j];
                    end
                end

            end
        end
    end

    point1 = point1 + [469 484];
    point2 = point2 + [469 484];
    point3 = point3 + [469 484];
    point4 = point4 + [469 484];
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    % pointi(1) = y and pointi(2) = x
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    point1(2) = point1(2)-imageXC;
    point2(2) = point2(2)-imageXC;
    point3(2) = point3(2)-imageXC;
    point4(2) = point4(2)-imageXC;

    point1(1) = imageYC -point1(1);
    point2(1) = imageYC -point2(1);
    point3(1) = imageYC -point3(1);
    point4(1) = imageYC -point4(1);

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    % point1(2)~x1 & point1(1)~y1 & point2(2)~x2 & point2(1)~y2 ......etc
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    %%fprintf(fid,'%.2f %10.2f %10.2f %10.2f %10.2f %10.2f %10.2f %10.2f\n', point1(2),
point1(1), point2(2), point2(1), point3(2), point3(1), point4(2), point4(1));
    fprintf(fid,'%.2f,%.2f,%.2f,%.2f,%.2f,%.2f,%.2f,%.2f\n', point4(2), point4(1), point1(2),
point1(1), point2(2), point2(1), point3(2), point3(1));

end
fclose(fid);
```

Appendix B Altered Pose Estimation C Sharp Code for Coplanar points

```
// 3D Pose Estimation (2) sample application
// AForge.NET Framework
// http://www.aforgenet.com/framework/
//
// Copyright © AForge.NET, 2009-2011
// contacts@aforgenet.com
// Changes made for CSIR Project by Jatin Harribhai (JH) and Johan Kieviet (JK)

using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Drawing;
using System.Text;
using System.Windows.Forms;
using System.Globalization;
using System.Reflection;
using System.IO;

using AForge;
using AForge.Math;
using AForge.Math.Geometry;

namespace PoseEstimation
{
    public partial class MainForm : Form
    {
        private readonly AForge.Point emptyPoint = new AForge.Point( -30000, -30000 );

        // image point of the object to estimate pose for
        private AForge.Point[] imagePoints = new AForge.Point[4];
        // model points
        private Vector3[] modelPoints = new Vector3[4];
        // camera's focal length
        private float focalLength;
        // estimated transformation
        private Matrix3x3 rotationMatrix, bestRotationMatrix, alternateRotationMatrix;
        private Vector3 translationVector, bestTranslationVector, alternateTranslationVector;
        private bool isPoseEstimated = false;
        private float modelRadius;

        // size of the opened image
        private Size imageSize = new Size( 0, 0 );

        // colors used to highlight points on image
        private Color[] pointsColors = new Color[4]
        {
            Color.Yellow, Color.Blue, Color.Red, Color.Lime
        };

        private bool useCoplanarPosit = true;

        // point index currently locating with mouse
        private int pointIndexToLocate = -1;
        private AForge.Point pointPreviousValue;

        // model used to draw coordinate system's axes
        private Vector3[] axesModel = new Vector3[]
        {
            new Vector3( 0, 0, 0 ),
            new Vector3( 1, 0, 0 ),
            new Vector3( 0, 1, 0 ),
            new Vector3( 0, 0, 1 ),
        };

        // Function for creating iterative file names for R and T matrix output files // added by JK
        private string getNextFileName(string fileName)
        {
            string extension = Path.GetExtension(fileName);

            int i = 0;
            while (File.Exists(fileName))
            {
```

```
                if (i == 0)
                    fileName = fileName.Replace(extension, "(" + ++i + ")" + extension);
                else
                    fileName = fileName.Replace("(" + i + ")" + extension, "(" + ++i + ")" +
extension);
            }

            return fileName;
        }
        // ----------------------------------------end JK


        // a structure describing built-in sample
        private struct Sample
        {
            public readonly string ImageName;
            public readonly AForge.Point[] ImagePoints;
            public readonly Vector3[] ModelPoints;
            public readonly float FocalLength;
            public readonly bool IsCoplanar;

            public Sample( string imageName, AForge.Point[] imagePoints, Vector3[] modelPoints, float
focalLength, bool isCoplanar )
            {
                ImageName   = imageName;
                ImagePoints = imagePoints;
                ModelPoints = modelPoints;
                FocalLength = focalLength;
                IsCoplanar  = isCoplanar;
            }
        }

        #region Embedded samples
        private Sample[] samples = new Sample[]
        {
            // -----
            new Sample(
                "sample1.jpg",
                new AForge.Point[]
                {
                    new AForge.Point( -4, 29 ),
                    new AForge.Point( -180, 86 ),
                    new AForge.Point( -5, -102 ),
                    new AForge.Point( 76, 137 ),
                },
                new Vector3[]
                {
                    new Vector3(  28,  28, -28 ),
                    new Vector3( -28,  28, -28 ),
                    new Vector3(  28, -28, -28 ),
                    new Vector3(  28,  28,  28 ),
                },
                640, false ),

            // -----
            new Sample(
                "sample2.jpg",
                new AForge.Point[]
                {
                    new AForge.Point( -11, 58 ),
                    new AForge.Point( -125, 84 ),
                    new AForge.Point( -7, -35 ),
                    new AForge.Point( 37, 124 ),
                },
                new Vector3[]
                {
                    new Vector3(  28,  28, -28 ),
                    new Vector3( -28,  28, -28 ),
                    new Vector3(  28, -28, -28 ),
                    new Vector3(  28,  28,  28 ),
                },
                640, false ),

            // -----
            new Sample(
                "sample3.jpg",
```

```
            new AForge.Point[]
            {
                new AForge.Point( 4, 55 ),
                new AForge.Point( -80, 81 ),
                new AForge.Point( 3, -8 ),
                new AForge.Point( 40, 109 ),
            },
            new Vector3[]
            {
                new Vector3(  28,  28, -28 ),
                new Vector3( -28,  28, -28 ),
                new Vector3(  28, -28, -28 ),
                new Vector3(  28,  28,  28 ),
            },
            640, false ),

        // -----
        new Sample(
            "sample4.jpg",
            new AForge.Point[]
            {
                new AForge.Point( -77, 48 ),
                new AForge.Point( 44, 66 ),
                new AForge.Point( 75, -36 ),
                new AForge.Point( -61, -58 ),
            },
            new Vector3[]
            {
                new Vector3( -56.5f, 0,  56.5f ),
                new Vector3(  56.5f, 0,  56.5f ),
                new Vector3(  56.5f, 0, -56.5f ),
                new Vector3( -56.5f, 0, -56.5f ),
            },
            640, true ),

        // -----
        new Sample(
            "sample5.jpg",
            new AForge.Point[]
            {
                new AForge.Point( -117, 33 ),
                new AForge.Point( -15, 86 ),
                new AForge.Point( 89, 38 ),
                new AForge.Point( -13, -30 ),
            },
            new Vector3[]
            {
                new Vector3( -56.5f, 0,  56.5f ),
                new Vector3(  56.5f, 0,  56.5f ),
                new Vector3(  56.5f, 0, -56.5f ),
                new Vector3( -56.5f, 0, -56.5f ),
            },
            640, true ),
    };
    #endregion

    // Class constructor
    public MainForm( )
    {
        InitializeComponent( );
        EnableControls( false );
        UpdatePictureBoxPositon( );

        imagePoint1ColorLabel.BackColor = pointsColors[0];
        imagePoint2ColorLabel.BackColor = pointsColors[1];
        imagePoint3ColorLabel.BackColor = pointsColors[2];
        imagePoint4ColorLabel.BackColor = pointsColors[3];

        if ( useCoplanarPosit )
        {
            copositRadio.Checked = true;
        }
        else
        {
            positRadio.Checked = true;
        }
```

```
        imagePoints[0] = emptyPoint;
        imagePoints[1] = emptyPoint;
        imagePoints[2] = emptyPoint;
        imagePoints[3] = emptyPoint;

        ClearEstimation( );
    }

    // On File->Exit - close the application
    private void exitToolStripMenuItem_Click( object sender, EventArgs e )
    {
        this.Close( );
    }

    // On File->Open - open an image file
    private void openImageToolStripMenuItem_Click( object sender, EventArgs e )
    {
        if ( openFileDialog.ShowDialog( ) == DialogResult.OK )
        {
            try
            {
                OpenImage( (Bitmap) Bitmap.FromFile( openFileDialog.FileName ) );

                // reset image points
                imagePoints[0] = emptyPoint;
                imagePoints[1] = emptyPoint;
                imagePoints[2] = emptyPoint;
                imagePoints[3] = emptyPoint;

                imagePoint1Box.Text =
                imagePoint2Box.Text =
                imagePoint3Box.Text =
                imagePoint4Box.Text = string.Empty;



                // set default focal length to image width
                //focalLength = imageSize.Width;
                //focalLength = 3859;   // AW100 385                                    //JK
                focalLength = 3844;   // AW110                                          //JK
                focalLengthBox.Text = focalLength.ToString( CultureInfo.InvariantCulture );


                //TextWriter tsw = new StreamWriter("Output.txt");                      //JK
                //var reader = new StreamReader("pixel points.txt");                    //JK
                //TextWriter tsm = new StreamWriter("Matrices.txt");                    //JK
    ////////////////////////////
    /////////// START //////////
    ////////////////////////////

                    using (var reader = new StreamReader("pixel points.txt"))
                    using (reader)
                    {
                        string line;
                        while ((line = reader.ReadLine()) != null)//reader.Readline added by JH

                        {
                            //
                            Console.WriteLine(line);
                            Console.ReadLine();
                            //tsw.WriteLine(line);

                            string[] bits = line.Split(',');                            // JH
                            //float z = Convert.ToSingle(bits[0]);
                            //imagePoints[0].X = z;
                            int i = 0;
                            for (int j = 0; j < bits.Length; j++)                       // JH
                            {
                                float z = Convert.ToSingle(bits[j]);                    // JH
                                // saving x                                             // JH
                                imagePoints[i].X = z;
                                j = j + 1;
                                // saving y                                             // JH
                                z = Convert.ToSingle(bits[j]);
                                imagePoints[i].Y = z;
```

```
                            // going to the next point                              // JH

                            i = i + 1;
                        }

                        EstimatePose();
                        ClearEstimation();
                    }
                }

////////////////////////////
////////// END  ////////////
////////////////////////////
            }
            catch ( Exception ex )
            {
                MessageBox.Show( "Failed opening selected file.\n\nException: " + ex.Message );
            }
        }
    }


    // Open one of the embedded samples
    private void openSampleToolStripMenuItem_Click( object sender, EventArgs e )
    {
        ToolStripMenuItem menuItem = (ToolStripMenuItem) sender;
        int sampleIndex = int.Parse( (string) menuItem.Tag );

        Sample sample = samples[sampleIndex];

        OpenEmbeddedImage( sample.ImageName );

        // set image points
        imagePoints = (AForge.Point[]) sample.ImagePoints.Clone( );

        imagePoint1Box.Text = imagePoints[0].ToString( );
        imagePoint2Box.Text = imagePoints[1].ToString( );
        imagePoint3Box.Text = imagePoints[2].ToString( );
        imagePoint4Box.Text = imagePoints[3].ToString( );

        // set model points
        //modelPoints = (Vector3[]) sample.ModelPoints.Clone( );

        modelPoint1xBox.Text = modelPoints[0].X.ToString( );
        modelPoint1yBox.Text = modelPoints[0].Y.ToString( );
        modelPoint1zBox.Text = modelPoints[0].Z.ToString( );

        modelPoint2xBox.Text = modelPoints[1].X.ToString( );
        modelPoint2yBox.Text = modelPoints[1].Y.ToString( );
        modelPoint2zBox.Text = modelPoints[1].Z.ToString( );

        modelPoint3xBox.Text = modelPoints[2].X.ToString( );
        modelPoint3yBox.Text = modelPoints[2].Y.ToString( );
        modelPoint3zBox.Text = modelPoints[2].Z.ToString( );

        modelPoint4xBox.Text = modelPoints[3].X.ToString( );
        modelPoint4yBox.Text = modelPoints[3].Y.ToString( );
        modelPoint4zBox.Text = modelPoints[3].Z.ToString( );

        // set focal length
        focalLength = sample.FocalLength;
        focalLengthBox.Text = focalLength.ToString( );

        // POSIT or Coplanar POSIT
        useCoplanarPosit = sample.IsCoplanar;
        if ( useCoplanarPosit )
        {
            copositRadio.Checked = true;
        }
        else
        {
            positRadio.Checked = true;
        }

        errorProvider.Clear( );
```

```
            EstimatePose();
        }

        // Enable/disable controls which are available when image is opened
        private void EnableControls( bool enable )
        {
            imagePointsGroupBox.Enabled = enable;
            modelPointsGroupBox.Enabled = enable;
            poseGroupBox.Enabled = enable;
        }

        // Close current image
        private void CloseImage( )
        {
            pictureBox.Image = null;
            EnableControls( false );
        }

        // Open image which is embedded into the assembly as resource
        private void OpenEmbeddedImage( string imageName )
        {
            // load arrow bitmap
            Assembly assembly = this.GetType( ).Assembly;
            Bitmap image = new Bitmap( assembly.GetManifestResourceStream( "PoseEstimation.Samples." +
imageName ) );
            OpenImage( image );
        }

        // Opens the specified image
        private void OpenImage( Bitmap image )
        {
            // close previous image if any
            CloseImage( );

            // open new image
            imageSize = image.Size;

            pictureBox.Image = image;
            pictureBox.Size = new Size( imageSize.Width + 2, imageSize.Height + 2 );
            imageSizeLabel.Text = string.Format( "{0} x {1}", image.Width, image.Height );

            ClearEstimation( );
            UpdatePictureBoxPositon( );
            EnableControls( true );
        }

        private void ClearEstimation( )
        {
            isPoseEstimated = false;
            estimatedTransformationMatrixControl.Clear( );
            bestPoseButton.Visible = false;
            alternatePoseButton.Visible = false;
        }

        // Paint image points on the image
        private void pictureBox_Paint( object sender, PaintEventArgs e )
        {
            Graphics g = e.Graphics;

            if ( pictureBox.Image != null )
            {
                int cx = imageSize.Width  / 2;
                int cy = imageSize.Height / 2;

                for ( int i = 0; i < 4; i++ )
                {
                    if ( imagePoints[i] != emptyPoint )
                    {
                        using ( Brush brush = new SolidBrush( pointsColors[i] ) )
                        {
                            g.FillEllipse( brush, new Rectangle(
                                (int) ( cx + imagePoints[i].X - 3 ),
                                (int) ( cy - imagePoints[i].Y - 3 ),
                                7, 7 ) );
                        }
```

```
                }
            }

            if ( ( isPoseEstimated ) && ( pointIndexToLocate == -1 ) )
            {
                AForge.Point[] projectedAxes = PerformProjection( axesModel,
                    // create transformation matrix
                    Matrix4x4.CreateTranslation( translationVector ) *       // 3: translate
                    Matrix4x4.CreateFromRotation( rotationMatrix ) *         // 2: rotate
                    Matrix4x4.CreateDiagonal(
                        new Vector4( modelRadius, modelRadius, modelRadius, 1 ) ), // 1: scale
                    imageSize.Width
                );

                using ( Pen pen = new Pen( Color.Blue, 5 ) )
                {
                    g.DrawLine( pen,
                        cx + projectedAxes[0].X, cy - projectedAxes[0].Y,
                        cx + projectedAxes[1].X, cy - projectedAxes[1].Y );
                }

                using ( Pen pen = new Pen( Color.Red, 5 ) )
                {
                    g.DrawLine( pen,
                        cx + projectedAxes[0].X, cy - projectedAxes[0].Y,
                        cx + projectedAxes[2].X, cy - projectedAxes[2].Y );
                }

                using ( Pen pen = new Pen( Color.Lime, 5 ) )
                {
                    g.DrawLine( pen,
                        cx + projectedAxes[0].X, cy - projectedAxes[0].Y,
                        cx + projectedAxes[3].X, cy - projectedAxes[3].Y );
                }
            }
        }
    }

    private AForge.Point[] PerformProjection( Vector3[] model, Matrix4x4 transformationMatrix, int
viewSize )
    {
        AForge.Point[] projectedPoints = new AForge.Point[model.Length];

        for ( int i = 0; i < model.Length; i++ )
        {
            Vector3 scenePoint = ( transformationMatrix * model[i].ToVector4( ) ).ToVector3( );

            projectedPoints[i] = new AForge.Point(
                (int) ( scenePoint.X / scenePoint.Z * viewSize ),
                (int) ( scenePoint.Y / scenePoint.Z * viewSize ) );
        }

        return projectedPoints;
    }

    // Update position of picture box so it is centred in the main panel
    private void UpdatePictureBoxPositon( )
    {
        pictureBox.Location = new System.Drawing.Point(
            ( mainPanel.Width - pictureBox.Width ) / 2,
            ( mainPanel.Height - pictureBox.Height ) / 2 );
    }

    // On resize of main form
    private void MainForm_Resize( object sender, EventArgs e )
    {
        UpdatePictureBoxPositon( );
    }

    // One of the locate point button were clicked
    private void locatePointButton_Click( object sender, EventArgs e )
    {
        pictureBox.Capture = true;

        Button sourceButton = (Button) sender;
        pointIndexToLocate = int.Parse( (string) sourceButton.Tag );
```

```
            pointPreviousValue = imagePoints[pointIndexToLocate];
            imagePoints[pointIndexToLocate] = emptyPoint;

            statusLabel.Text = string.Format( "Locate point #{0} in the image ...", pointIndexToLocate
+ 1 );
            pictureBox.Invalidate( );
        }

        // Mouse click on the image - accept new point or reject it (depending on mouse button)
        private void pictureBox_MouseClick( object sender, MouseEventArgs e )
        {
            if ( pointIndexToLocate != -1 )
            {
                pictureBox.Cursor = Cursors.Default;
                pictureBox.Capture = false;
                statusLabel.Text = string.Empty;

                if ( e.Button == MouseButtons.Left )
                {
                    int x = Math.Max( 0, Math.Min( e.X, imageSize.Width - 1 ) );
                    int y = Math.Max( 0, Math.Min( e.Y, imageSize.Height - 1 ) );

                    imagePoints[pointIndexToLocate] = new AForge.Point( x - imageSize.Width / 2,
imageSize.Height / 2 - y );

                    TextBox imagePointTextBox = (TextBox) imagePointsGroupBox.Controls[string.Format(
"imagePoint{0}Box", pointIndexToLocate + 1 )];
                    imagePointTextBox.Text = imagePoints[pointIndexToLocate].ToString( );
                }
                else
                {
                    imagePoints[pointIndexToLocate] = pointPreviousValue;
                }

                ClearEstimation( );

                pointIndexToLocate = -1;
                pictureBox.Invalidate( );
            }
        }

        private void pictureBox_MouseMove( object sender, MouseEventArgs e )
        {
            if ( pointIndexToLocate != -1 )
            {
                pictureBox.Cursor = Cursors.Help;
            }
        }

        // Leaving one of the model point's boxes - validate it
        private void modelPointBox_Leave( object sender, EventArgs e )
        {
            GetCoordinateValue( (TextBox) sender );
        }

        private void GetCoordinateValue( TextBox textBox )
        {
            int tag = int.Parse( (string) textBox.Tag );
            int pointIndex = tag / 10;
            int coordinateIndex = tag % 10;
            float coordinateValue, oldValue = 0;

            textBox.Text = textBox.Text.Trim( );

            // try parsing the coordinate value
            if ( float.TryParse( textBox.Text, NumberStyles.Float, CultureInfo.InvariantCulture, out
coordinateValue) )
            {
                switch ( coordinateIndex )
                {
                    case 0:
                        oldValue = modelPoints[pointIndex].X;
                        modelPoints[pointIndex].X = coordinateValue;
                        break;
                    case 1:
```

```
                        oldValue = modelPoints[pointIndex].Y;
                        modelPoints[pointIndex].Y = coordinateValue;
                        break;
                    case 2:
                        oldValue = modelPoints[pointIndex].Z;
                        modelPoints[pointIndex].Z = coordinateValue;
                        break;
                }
                errorProvider.Clear( );

                if ( oldValue != coordinateValue )
                {
                    ClearEstimation( );
                }
            }
            else
            {
                Label pointLabel = (Label) modelPointsGroupBox.Controls[string.Format(
"modelPoint{0}Label", pointIndex + 1 )];

                errorProvider.SetError( pointLabel, string.Format( "Failed parsing {0} coordinate",
                    ( coordinateIndex == 0 ) ? "X" : ( ( coordinateIndex == 1 ) ? "Y" : "Z" ) ) );

                textBox.Text = string.Empty;
            }
        }

        // Validate focal length on leaving the text box
        private void focalLengthBox_Leave( object sender, EventArgs e )
        {
            float value;

            if ( float.TryParse( focalLengthBox.Text, NumberStyles.Float,
CultureInfo.InvariantCulture, out value ) )
            {
                focalLength = value;
            }
            else
            {
                focalLengthBox.Text = focalLength.ToString( );
                errorProvider.SetError( focalLengthLabel, "Wrong focal length was specified. Restored
to previous value." );
            }
        }

        // Switch between POSIT and CoPOSIT algorithms
        private void copositRadio_CheckedChanged( object sender, EventArgs e )
        {
            useCoplanarPosit = copositRadio.Checked;
        }

        private void estimatePostButton_Click( object sender, EventArgs e )
        {
            EstimatePose();
        }

        // Estimate 3D position
        private void EstimatePose()
        {
            try
            {


                imagePoint1Box.Text = imagePoints[0].ToString();
                imagePoint2Box.Text = imagePoints[1].ToString();
                imagePoint3Box.Text = imagePoints[2].ToString();
                imagePoint4Box.Text = imagePoints[3].ToString();

                // check if all image coordinates are specified
                if ( ( string.IsNullOrEmpty( imagePoint1Box.Text ) ) ||
                     ( string.IsNullOrEmpty( imagePoint2Box.Text ) ) ||
                     ( string.IsNullOrEmpty( imagePoint3Box.Text ) ) ||
                     ( string.IsNullOrEmpty( imagePoint4Box.Text ) ) )
                {
                    throw new ApplicationException( "Some image coordinates are not specified." );
                }
```

```
//Model points being hard coded---------------- // Section added by JH
modelPoints[0].X = 257;
modelPoints[0].Y = 0;
modelPoints[0].Z = 506;

modelPoints[1].X = 257;
modelPoints[1].Y = 0;
modelPoints[1].Z = -506;

modelPoints[2].X = -257;
modelPoints[2].Y = 0;
modelPoints[2].Z = -506;

modelPoints[3].X = -257;
modelPoints[3].Y = 0;
modelPoints[3].Z = 506;
//---------------------------end JH

modelPoint1xBox.Text = modelPoints[0].X.ToString();
modelPoint1yBox.Text = modelPoints[0].Y.ToString();
modelPoint1zBox.Text = modelPoints[0].Z.ToString();

modelPoint2xBox.Text = modelPoints[1].X.ToString();
modelPoint2yBox.Text = modelPoints[1].Y.ToString();
modelPoint2zBox.Text = modelPoints[1].Z.ToString();

modelPoint3xBox.Text = modelPoints[2].X.ToString();
modelPoint3yBox.Text = modelPoints[2].Y.ToString();
modelPoint3zBox.Text = modelPoints[2].Z.ToString();

modelPoint4xBox.Text = modelPoints[3].X.ToString();
modelPoint4yBox.Text = modelPoints[3].Y.ToString();
modelPoint4zBox.Text = modelPoints[3].Z.ToString();

// check if all model coordinates are specified
if ( ( string.IsNullOrEmpty( modelPoint1xBox.Text ) ) ||
     ( string.IsNullOrEmpty( modelPoint2xBox.Text ) ) ||
     ( string.IsNullOrEmpty( modelPoint3xBox.Text ) ) ||
     ( string.IsNullOrEmpty( modelPoint4xBox.Text ) ) ||
     ( string.IsNullOrEmpty( modelPoint1yBox.Text ) ) ||
     ( string.IsNullOrEmpty( modelPoint2yBox.Text ) ) ||
     ( string.IsNullOrEmpty( modelPoint3yBox.Text ) ) ||
     ( string.IsNullOrEmpty( modelPoint4yBox.Text ) ) ||
     ( ( !useCoplanarPosit ) && (
       ( string.IsNullOrEmpty( modelPoint1zBox.Text ) ) ||
       ( string.IsNullOrEmpty( modelPoint2zBox.Text ) ) ||
       ( string.IsNullOrEmpty( modelPoint3zBox.Text ) ) ||
       ( string.IsNullOrEmpty( modelPoint4zBox.Text ) ) ) ) )
{
    throw new ApplicationException( "Some model coordinates are not specified." );
}

useCoplanarPosit = true;

// calculate model's centre
Vector3 modelCenter = new Vector3(
    ( modelPoints[0].X + modelPoints[1].X + modelPoints[2].X + modelPoints[3].X ) / 4,
    ( modelPoints[0].Y + modelPoints[1].Y + modelPoints[2].Y + modelPoints[3].Y ) / 4,
    ( modelPoints[0].Z + modelPoints[1].Z + modelPoints[2].Z + modelPoints[3].Z ) / 4
);

// calculate ~ model's radius
modelRadius = 0;
foreach ( Vector3 modelPoint in modelPoints )
{
    float distanceToCenter = ( modelPoint - modelCenter ).Norm;
    if ( distanceToCenter > modelRadius )
    {
        modelRadius = distanceToCenter;
    }
}

if ( !useCoplanarPosit )
{
    Posit posit = new Posit( modelPoints, focalLength );
    posit.EstimatePose( imagePoints, out rotationMatrix, out translationVector );
```

```
                bestPoseButton.Visible = alternatePoseButton.Visible = false;
            }
            else
            {
                CoplanarPosit coposit = new CoplanarPosit( modelPoints, focalLength );
                coposit.EstimatePose( imagePoints, out rotationMatrix, out translationVector );

                bestRotationMatrix = coposit.BestEstimatedRotation;
                bestTranslationVector = coposit.BestEstimatedTranslation;

                alternateRotationMatrix = coposit.AlternateEstimatedRotation;
                alternateTranslationVector = coposit.AlternateEstimatedTranslation;

                //R and T matrices writen to .txt file  ------------- //Section added by JH
                TextWriter tsm = new StreamWriter(getNextFileName("Matrices.txt"));
                tsm.Write(bestRotationMatrix.V00);
                tsm.Write(" ");
                tsm.Write(bestRotationMatrix.V01);
                tsm.Write(" ");
                tsm.WriteLine(bestRotationMatrix.V02);
                tsm.Write(bestRotationMatrix.V10);
                tsm.Write(" ");
                tsm.Write(bestRotationMatrix.V11);
                tsm.Write(" ");
                tsm.WriteLine(bestRotationMatrix.V12);
                tsm.Write(bestRotationMatrix.V20);
                tsm.Write(" ");
                tsm.Write(bestRotationMatrix.V21);
                tsm.Write(" ");
                tsm.WriteLine(bestRotationMatrix.V22);
                tsm.WriteLine(bestTranslationVector);
                tsm.Close();
                //--------------------------end JH
                bestPoseButton.Visible = alternatePoseButton.Visible = true;

            }

            isPoseEstimated = true;
            UpdateEstimationInformation( );
            pictureBox.Invalidate( );
        }
        catch ( ApplicationException ex )
        {
            MessageBox.Show( ex.Message, "Error", MessageBoxButtons.OK, MessageBoxIcon.Error );
        }
    }

    private void UpdateEstimationInformation( )
    {
        estimatedTransformationMatrixControl.SetMatrix(
            Matrix4x4.CreateTranslation( translationVector ) *
            Matrix4x4.CreateFromRotation( rotationMatrix ) );

        float estimatedYaw;
        float estimatedPitch;
        float estimatedRoll;

        rotationMatrix.ExtractYawPitchRoll( out estimatedYaw, out estimatedPitch, out
estimatedRoll );

        estimatedYaw   *= (float) ( 180.0 / Math.PI );
        estimatedPitch *= (float) ( 180.0 / Math.PI );
        estimatedRoll  *= (float) ( 180.0 / Math.PI );

        estimationLabel.Text = string.Format( "Rotation: (yaw(y)={0}, pitch(x)={1}, roll(z)={2})",
            estimatedYaw, estimatedPitch, estimatedRoll );
    }

    // Select best pose estimation
    private void bestPoseButton_Click( object sender, EventArgs e )
    {
        rotationMatrix = bestRotationMatrix;
        translationVector = bestTranslationVector;

        UpdateEstimationInformation( );
```

```
            pictureBox.Invalidate( );
        }

        // Select alternate pose estimation
        private void alternatePoseButton_Click( object sender, EventArgs e )
        {
            rotationMatrix = alternateRotationMatrix;
            translationVector = alternateTranslationVector;

            UpdateEstimationInformation( );
            pictureBox.Invalidate( );
        }
    }

}
```

Appendix C Image Processing Script for the 3D Object Tracking

```
%These scripts are project specific and calculates the image coordinates for
%the 3D Object to be tracked in the thesis work of Johan Kieviet
%
%Copyright © CSIR 2013, program by Jatin Harribhai (JH) and Johan Kieviet (JK)
%



fid = fopen('pixel points.txt','w');
 beginn=15824;
 endd=17982;
 imageXC= 2304;
 imageYC= 1728;
 focalLength = 3844;
 objectPoints = [-100 100 100;-100 -100 100;100 100 100;-100 100 -100];

 for k = beginn:endd                              %nzFrames
    filename = sprintf('Pic%d.jpg',k);            %added by JK

    imagergb = imread(filename,'jpg');            %changed by JK

    imageb = imagergb(:,:,3);                     %Image3 is only BLUE layer of grey image

    image3 = imcrop(imageb,[1085 453 312 297]); %% Starboard side "Nikon AW100"
    %%--Converting RGB image to BW image NB! Specifies intensities to keep
    %%-----------------------------------------------------------------------
    bwimage2 = im2bw(image3, 0.6);

    %%--Eroding and Dilating of BW image sufficient times
    %%-----------------------------------------------------------------------
    se1 = strel('disk',1);
    erodeBW = imerode(bwimage2,se1);

    se2 = strel('disk',1);
    dilateBW = imdilate(erodeBW,se2);

    cc = bwconncomp(dilateBW,4);

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    %% part 1 side view
    part1 = false(size(dilateBW));
    part1(cc.PixelIdxList{1}) = true;
%     figure
%     imshow(part1)

    [row,col] = size(part1);
    low_row = row;
    high_row = 1;
    low_col = col;
    high_col = 1;

    for i = 1:row
        for j = 1:col
            if (part1(i,j) == 1)
                if (i < low_row)
                    low_row = i;
                    point6 = [i j];
                else if ( i > high_row)
                        high_row = i;
                        point8 = [i j];
                    end
                end

                if (j < low_col)
                    low_col = j;
                    point5 = [i j];
                else if (j > high_col)
                        high_col = j;
                        point7 = [i j];
                    end
                end

            end
        end
    end
```

```
    %% part 2 top view
    part2 = false(size(dilateBW));
    part2(cc.PixelIdxList{2}) = true;

    [row,col] = size(part2);
    low_row = row;
    high_row = 1;
    low_col = col;
    high_col = 1;

    for i = 1:row
        for j = 1:col
            if (part2(i,j) == 1)
                if (i < low_row)
                    low_row = i;
                    point2 = [i j];
                else if ( i > high_row)
                        high_row = i;
                        point4 = [i j];
                    end
                end

                if (j < low_col)
                    low_col = j;
                    point1 = [i j];
                else if (j > high_col)
                        high_col = j;
                        point3 = [i j];
                    end
                end

            end
        end
    end


point1= ((point1+point6)/2) + [653 1085];
point2= point2 + [653 1085];
point3= point3 + [653 1085];
point4= ((point4+point7)/2) + [653 1085];
point8= point8 + [653 1085];
point5= point5 + [653 1085];



    point1(2) = point1(2)-imageXC;
    point2(2) = point2(2)-imageXC;
    point3(2) = point3(2)-imageXC;
    point4(2) = point4(2)-imageXC;
    point8(2) = point8(2)-imageXC;
    point5(2) = point5(2)-imageXC;

    point1(1) = imageYC -point1(1);
    point2(1) = imageYC -point2(1);
    point3(1) = imageYC -point3(1);
    point4(1) = imageYC -point4(1);
    point8(1) = imageYC -point8(1);
    point5(1) = imageYC -point5(1);

fprintf(fid,'%.2f,%.2f,%.2f,%.2f,%.2f,%.2f,%.2f,%.2f,%.2f,%.2f,%.2f,%.2f\n', point1(2),
point1(1), point2(2), point2(1), point3(2), point3(1), point4(2), point4(1), point8(2),
point8(1), point5(2), point5(1));


%%%%%%%%%%%%%%%%%%%%%%%%%Calculate Matrices and write to file%%%%%%%%%%%%%%%%
imagePoints = [point4(2) point4(1); point8(2) point8(1); point1(2), point1(1); point3(2)
point3(1)];
[Ro, Tr] = classicPosit(imagePoints, objectPoints, focalLength);

if k == beginn
    filename='Matrices.txt';
end

if k >= beginn+1
    name1='Matrices(';
    name2=strcat(name1, num2str(k-beginn));
    filename=strcat(name2, ').txt');
```

```
end

fid1 = fopen(filename,'w');
fprintf(fid1,'%d,%d,%d\n',Ro(1),Ro(4),Ro(7));
fprintf(fid1,'%d,%d,%d\n',Ro(2),Ro(5),Ro(8));
fprintf(fid1,'%d,%d,%d\n',Ro(3),Ro(6),Ro(9));
fprintf(fid1,'%d,%d,%d\n',Tr(1),Tr(2),Tr(3));

fclose(fid1);

end
fclose(fid);
```

Appendix D Matlab Script Converting Rotation and Translation Matrices to 6DOF for the LED and Object Tracking

```
% RT2SixDof Convert a Camera perspective Rotation and Translation Matrix to 6DOF for the % %
%  objects.
% Note: The input R and T must be already scaled and calculated with image coordinates, % %
%  object coordinates (mm) and focal length
%
% Also Note: This code is still in the development stage
% Input Transformation file is as follow:
%   r11 r12 r13     //Row 1 to 3 is space separated
%   r21 r22 r23
%   r23 r32 r33
%   X   ,Y  ,Z      //Row 4 is comma separated
%
% Copyright (C) 2014, CSIR Stellenbosch, by Johan Kieviet

function obj6dof = RT2SixDof()

close all;
clear all;
fid1 = fopen('JKLED_01B\SixDOF30hzModel.txt','w');                  %%CHECK <----------<<
fid2 = fopen('JKLED_01B\SixDOF30hzProto.txt','w');                  %%CHECK <----------<<

    dirlist = dir('JKLED_01B\Matrices*.txt');                       %%CHECK <----------<<

    for k = 0:length(dirlist)-1;
        %Reading in .txt files containing Transformation Matrix and creating
        %seperate Ri and Ti Matrices
        if k == 0
            Tr0 = load('JKLED_01B\Matrices.txt');                   %%CHECK <----------<<
            R0 = [Tr0(1) Tr0(5) Tr0(9);Tr0(2) Tr0(6) Tr0(10);Tr0(3) Tr0(7) Tr0(11)];
            T0 = [Tr0(4);Tr0(8);Tr0(12)];
        end

        if k >= 1
            filename = strcat('JKLED_01B\Matrices(', num2str(k));  %%CHECK <----------<<
            filename = strcat(filename, ').txt'); %% example
                                                  %%filename='JKLED_01S\Matrices(1).txt'
            Trtemp = load(filename);
            Rtemp = [Trtemp(1) Trtemp(5) Trtemp(9);Trtemp(2) Trtemp(6) Trtemp(10);Trtemp(3)
Trtemp(7) Trtemp(11)];
            Ttemp = [Trtemp(4);Trtemp(8);Trtemp(12)];
)

        end
        %Converting rotation matrix seen from camera perspective to the object
        %perspective. Everything is estimated with 'R0' as the zero starting
        %point
        if k == 0
            Ra0 = transpose(R0)*R0;         %setting zero rotation point
            Rtempa =Ra0;
        end

        if k >= 1
            Rtempa = transpose(R0)*Rtemp;
            %%eval(sprintf('Ra%d=Rtempa;',k)); %%Supressed (used in development)
        end

        %Converting translations seen from camera perspective to the object
        %perspective. Everything is estimated with 'T0' as the zero starting
        %point
        if k == 0
            Ta0 = Ra0*(transpose(R0))*(T0-T0);%setting zero translation point
            Ttempa = Ta0;
        end

        if k >= 1
            Ttempa = Rtempa*(transpose(Rtemp))*(Ttemp-T0);
        end

        %Calculating Rotations: Roll, Pitch and Yaw from object perspective

        RPY = tr2rpy(Rtempa, 'deg');

        %time (in model)
```

```
        if k==0
            time = 0;
        end
        time = time + 1/(29.97);

        %Printing to .txt file (SixDOF.txt)
        %columns are Time(s), Surge, Sway, Heave, Roll, Pitch, Yaw  (comma separated)


fprintf(fid1,'%.2f,%.2f,%.2f,%.2f,%.2f,%.2f,%.2f\n',time,Ttempa(1),Ttempa(2),Ttempa(3),RPY(1),
RPY(2),RPY(3));

fprintf(fid2,'%.2f,%.2f,%.2f,%.2f,%.2f,%.2f,%.2f\n',time*10,Ttempa(1)*100,Ttempa(2)*100,Ttempa
(3)*100,RPY(1),RPY(2),RPY(3));


    end

end
```

Appendix E Matlab Script Converting object bound 6DOF to vessel bound 6DOF

```
% This script does the following:
% 1)Scale and convert 6DOF from object to vessel coordinate system
% 2)Remove unnecessary spikes from data
% 3)Convert between Left hand and right hand coordinate systems
% 4)Some filtering and smoothing is also done
% 5)Printing the 6DOF to be compared to the Keoship results
%
% Also Note: This code is still in the development stage
%
% Copyright (C) 2014, CSIR Stellenbosch, by Johan Kieviet
% Plot figures
%
clear all;
  X= load('JKLED_01S\SixDOF30hzProto.txt');


  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
  %%%%Remove spikes above 3 deg from data%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

  for i = 1:length(X)
         if i>1

          if abs(X(i,5)-X(i-1,5))>=3    %KEO Roll
              X(i,5)=X(i-1,5);
          end
          if abs(X(i,6)-X(i-1,6))>=3    %KEO yaw
              X(i,6)=X(i-1,6);
          end
          if abs(X(i,7)-X(i-1,7))>=3    %KEO Pitch
              X(i,7)=X(i-1,7);
          end
        end
  end
  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
  %%%%%%%%%%% Converting left hand axes(x,z,y) to right hand axes(x,y,z)%%%
  XXL = X(:,2);
  YYL = X(:,3);
  ZZL = X(:,4);
  X(:,2) = XXL;
  X(:,3) = ZZL;
  X(:,4) = YYL;
  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
  %%%%%%%%% Converting left hand RPY (RYP) to right hand RPY (RPY)%%%%%%%%%
  RL = X(:,5);
  PL = X(:,6);
  YL = X(:,7);
  X(:,5)= -RL;
  X(:,6)= -YL;
  X(:,7)= -PL;

  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
  %%% Filter/smooth %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

  X(:,2) =medfilt1(X(:,2),4);
  X(:,3) =medfilt1(X(:,3),4);
  X(:,4) =medfilt1(X(:,4),4);
  X(:,5) =medfilt1(X(:,5),4);
  X(:,6) =medfilt1(X(:,6),4);
  X(:,7) =medfilt1(X(:,7),4);
  X = sgolayfilt(X,3,41);
  X = sgolayfilt(X,3,41);
  X = sgolayfilt(X,3,41);

  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
  %%%%%%%%% Translating from centre of object to CoG of vessel%%%%%%%%%%%%
  %Xobj=0; Yobj=0; Zobj=130; %most starboard block
  Xobj=71000; Yobj=0; Zobj=13000; %LED

  X(:,2)= X(:,2) + Yobj*degtorad(X(:,7)) - Zobj*degtorad(X(:,6));
  X(:,3)= X(:,3) - Xobj*degtorad(X(:,7)) + Zobj*degtorad(X(:,5));
  X(:,4)= X(:,4) + Xobj*degtorad(X(:,6)) - Yobj*degtorad(X(:,5));
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%% For PLOTS, convert second --> hours   & mm ---> m %%%%%%%%%%%%%
X(:,1)=X(:,1)/(60*60);
X(:,2)=X(:,2)/1000;
X(:,3)=X(:,3)/1000;
X(:,4)=-X(:,4)/1000;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%    PLOT    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

TLim= X(length(X),1);

fig = figure;
set(fig,'Position',[200 100 600 800])
subplot(6,1,1)
plot(X(:,1),X(:,2),'k')
ylabel('surge [m]')
pos = get(gca,'Position');
set(gca,'XLim',[0 TLim],'XTickLabel',[],'Position',pos)

subplot(6,1,2)
plot(X(:,1),X(:,3),'k')
ylabel('sway [m]')
set(gca,'XLim',[0 TLim],'XTickLabel',[])

subplot(6,1,3)
plot(X(:,1),X(:,4),'k')
ylabel('heave [m]')
set(gca,'XLim',[0 TLim],'XTickLabel',[])

subplot(6,1,4)
plot(X(:,1),X(:,5),'k')
ylabel('roll [deg]')
set(gca,'XLim',[0 TLim],'XTickLabel',[])

subplot(6,1,5)
plot(X(:,1),X(:,6),'k')
ylabel('pitch [deg]')
set(gca,'XLim',[0 TLim],'XTickLabel',[])

subplot(6,1,6)
plot(X(:,1),X(:,7),'k')
ylabel('yaw [deg]')
pos = get(gca,'Position');
xlabel('t [h]')
set(gca,'XLim',[0 TLim],'Position',pos)
```

Appendix F Illustration of Transient Point and Noise Removal on 2D LED Tracking Data

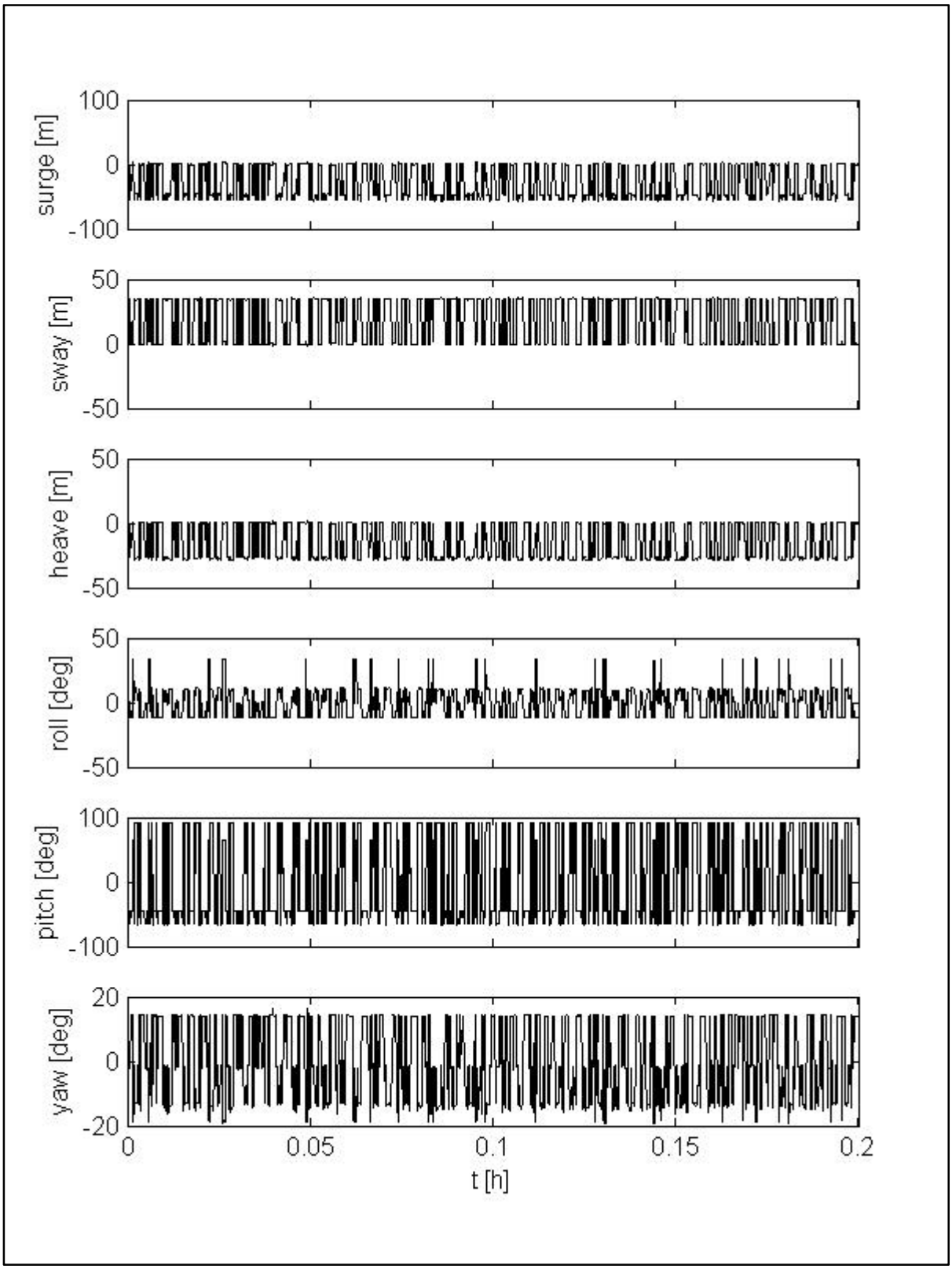

Figure F.1: Illustration of raw measurement data for Test LED02S

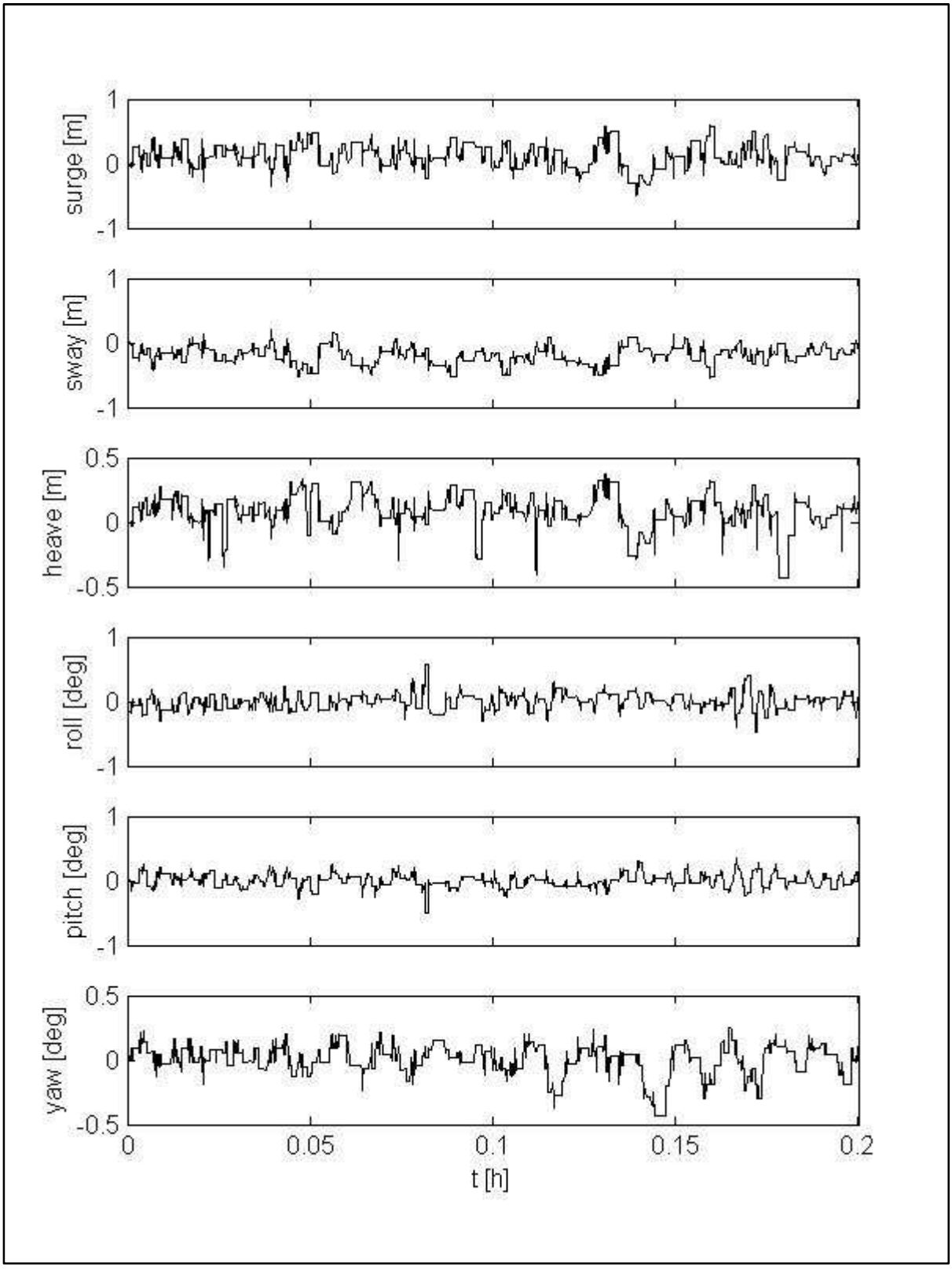

Figure F.2: Illustration of measurment data after removal of transients for Test LED02S

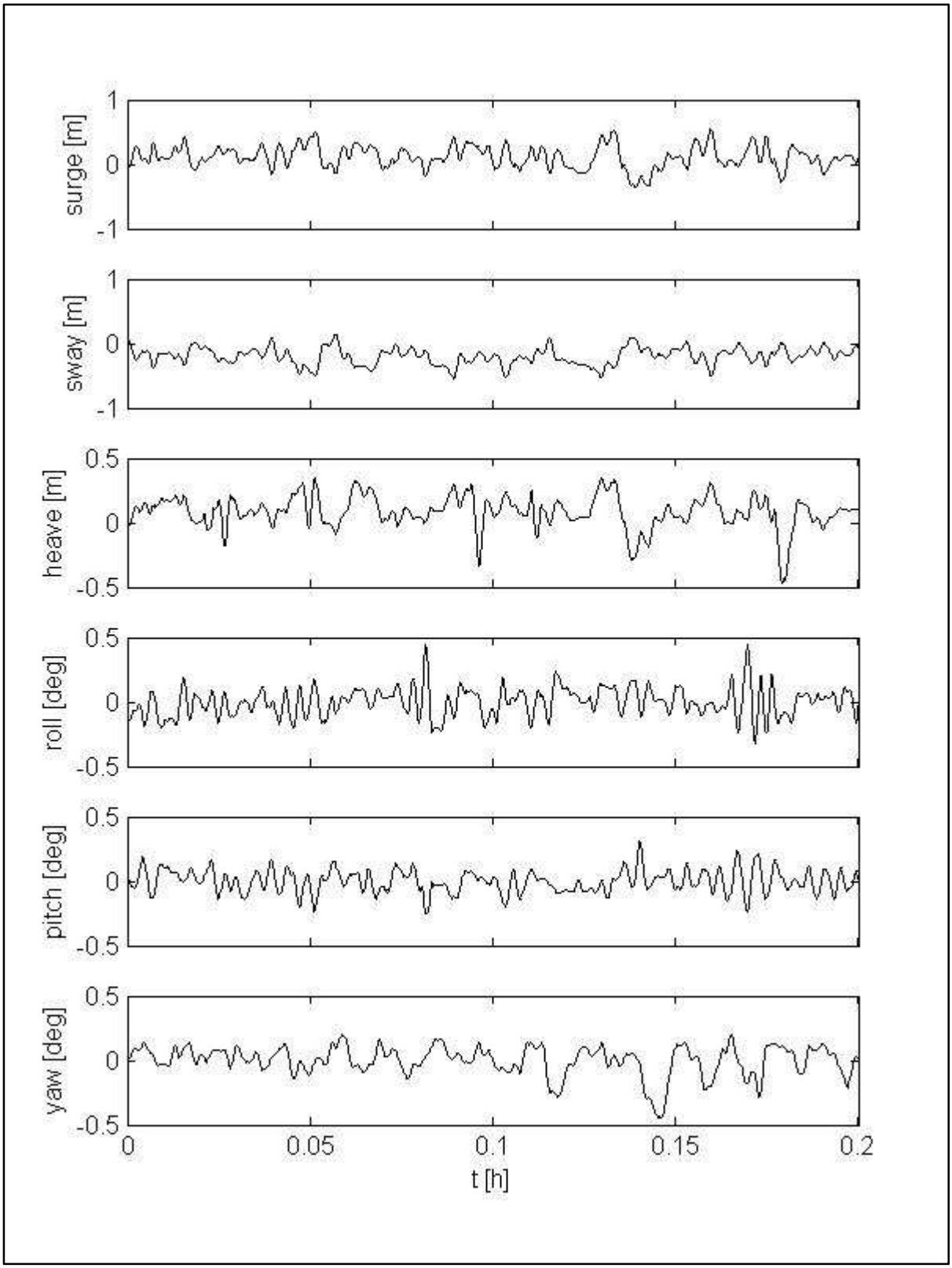

Figure F.3: Illustration of measurment data after removal of transients and high frequency noise for Test LED02S

Appendix G Motion Plots for Physical Model Tests

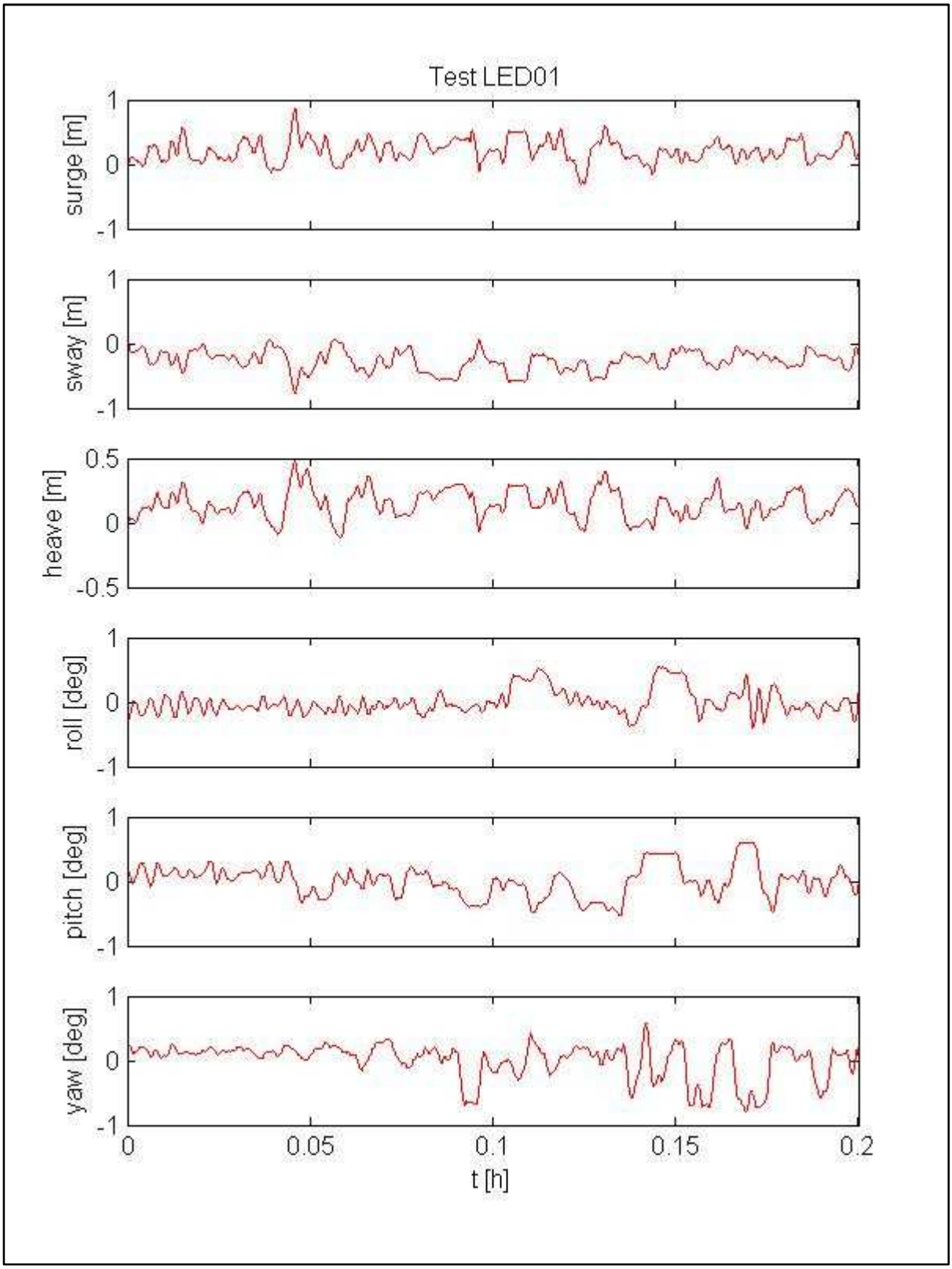

Figure G.1 : Motion results for tests LED01

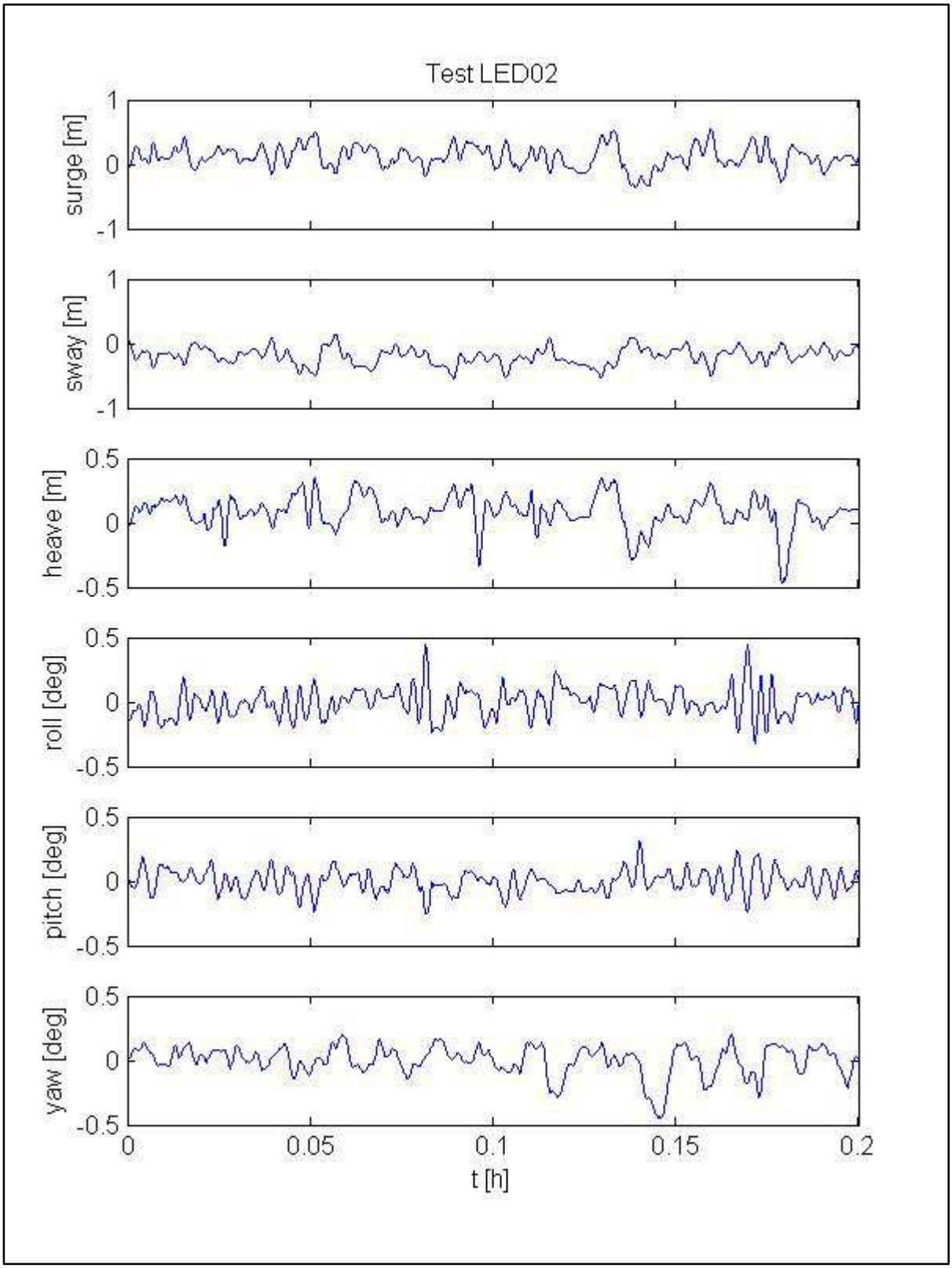

Figure G.2: Motion results for tests LED02

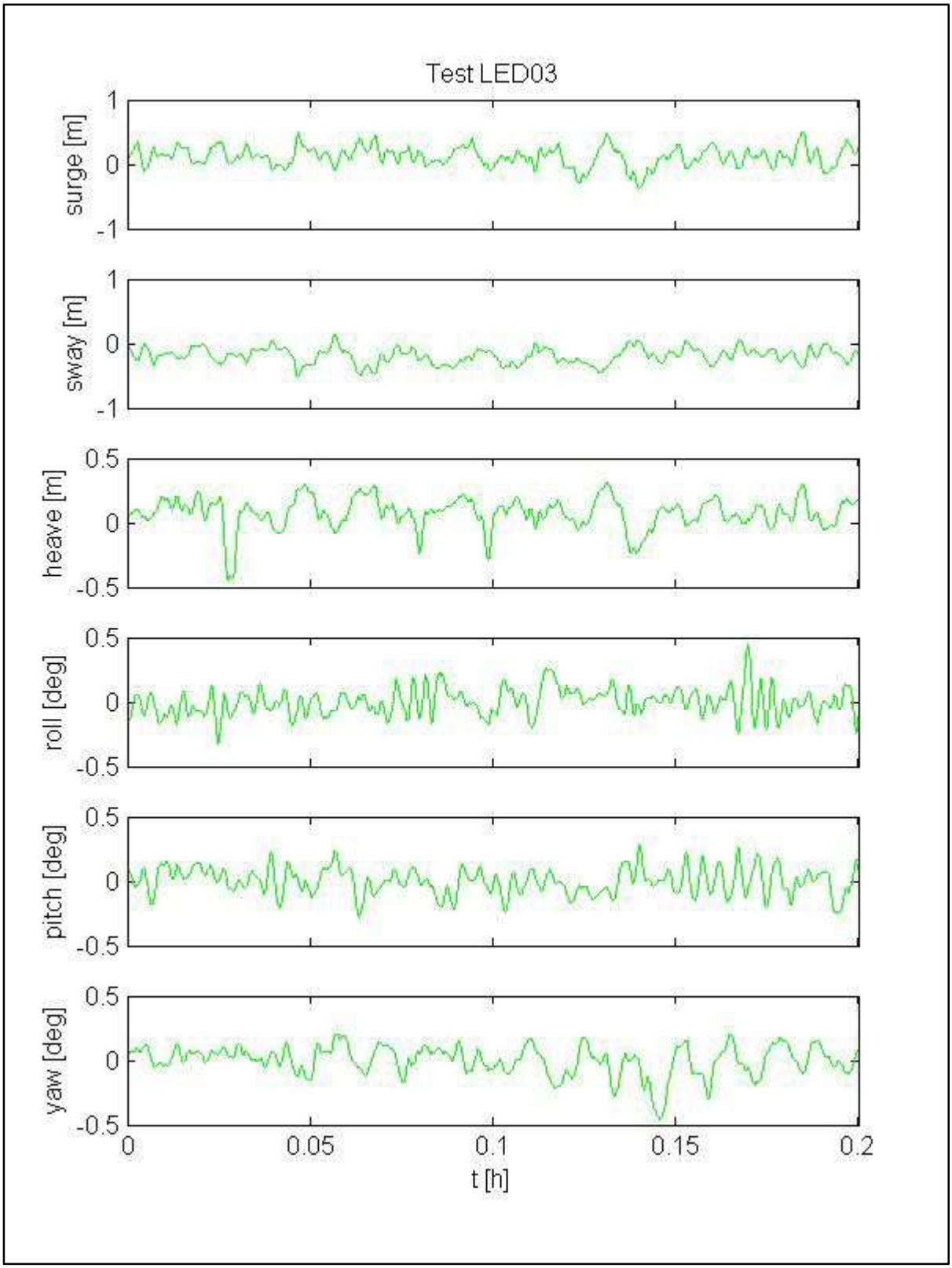

Figure G.3: Motion results for tests LED03

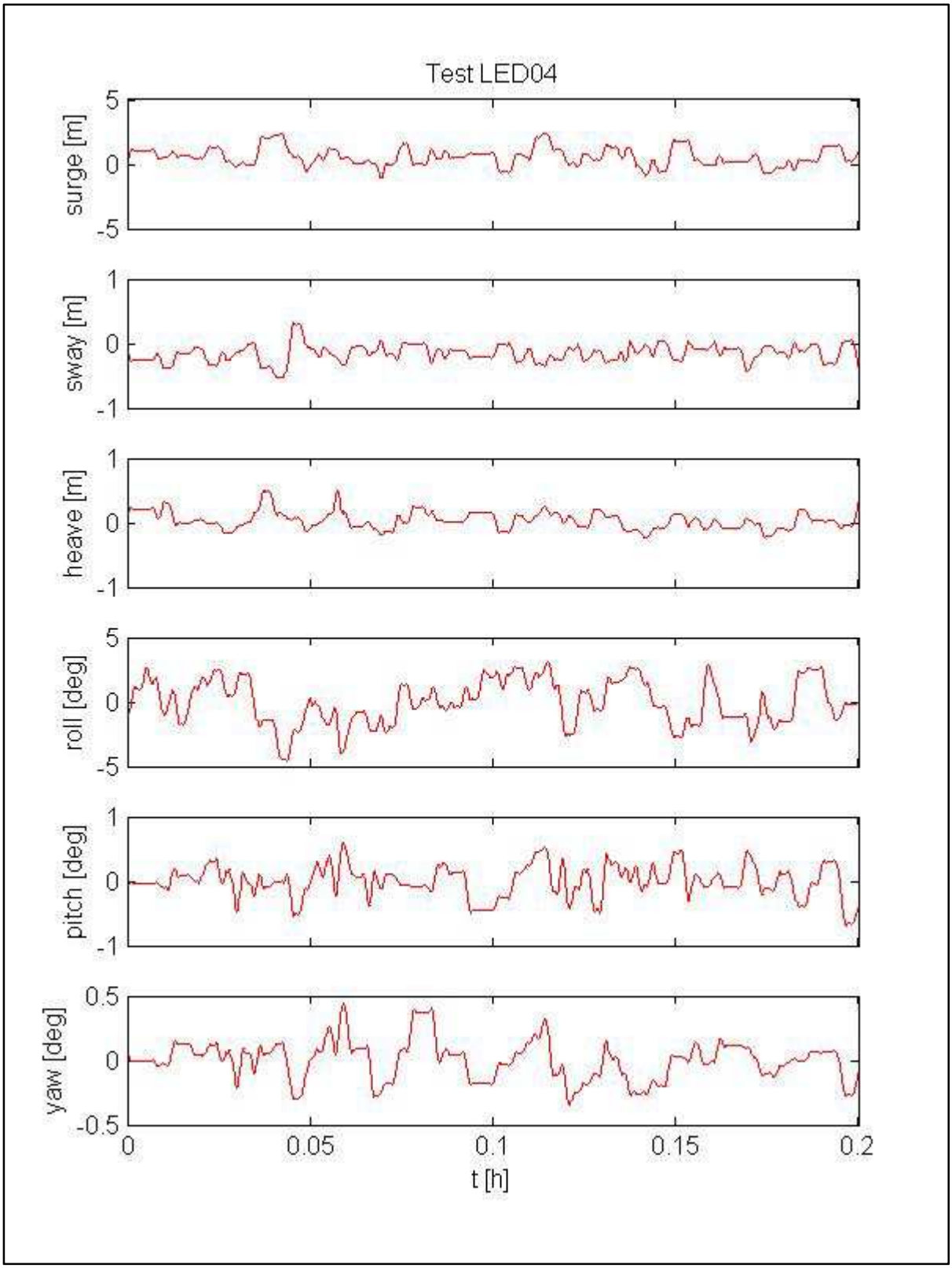

Figure G.4: Motion results for tests LED04

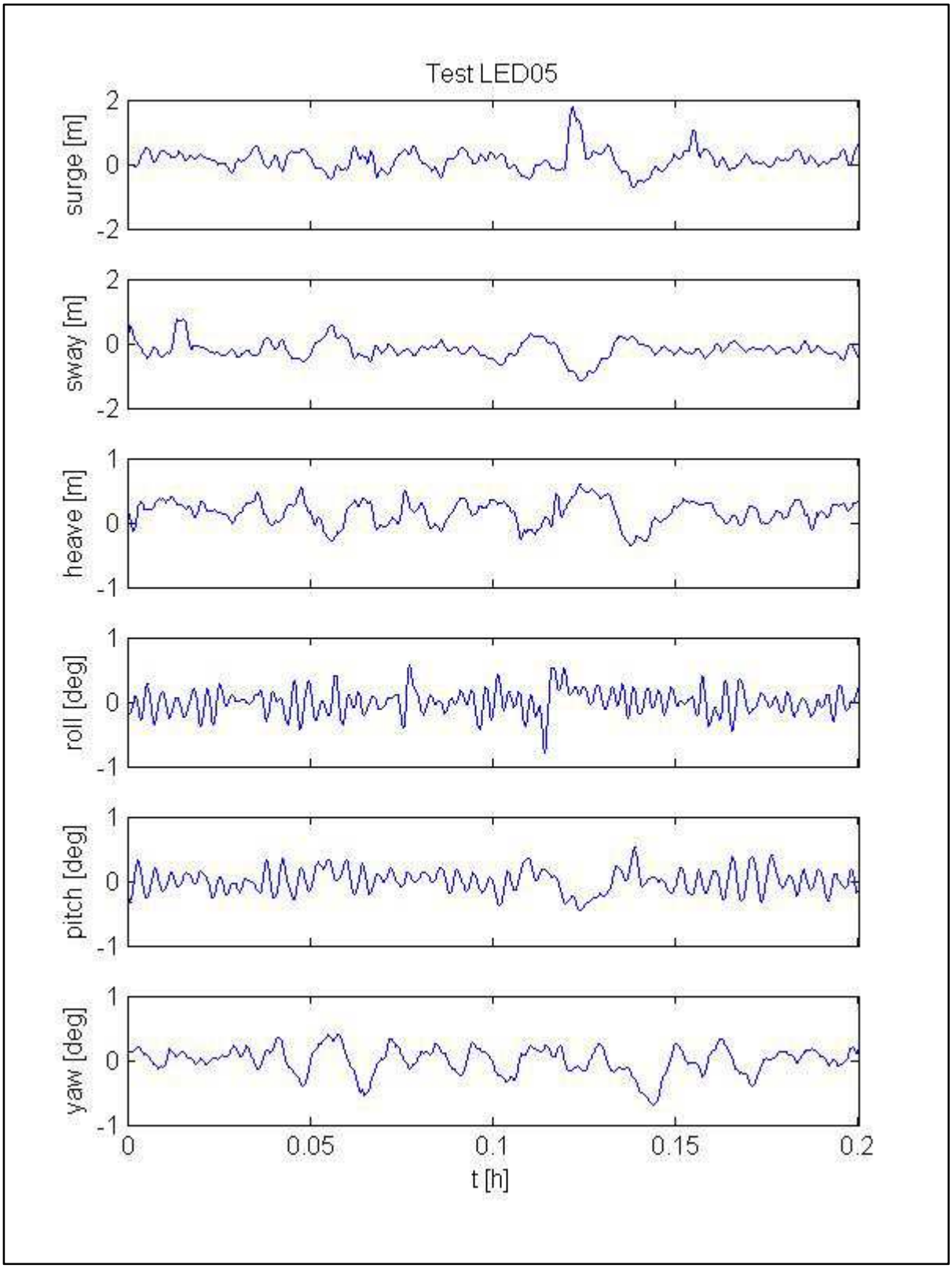

Figure G.5: Motion results for tests LED05

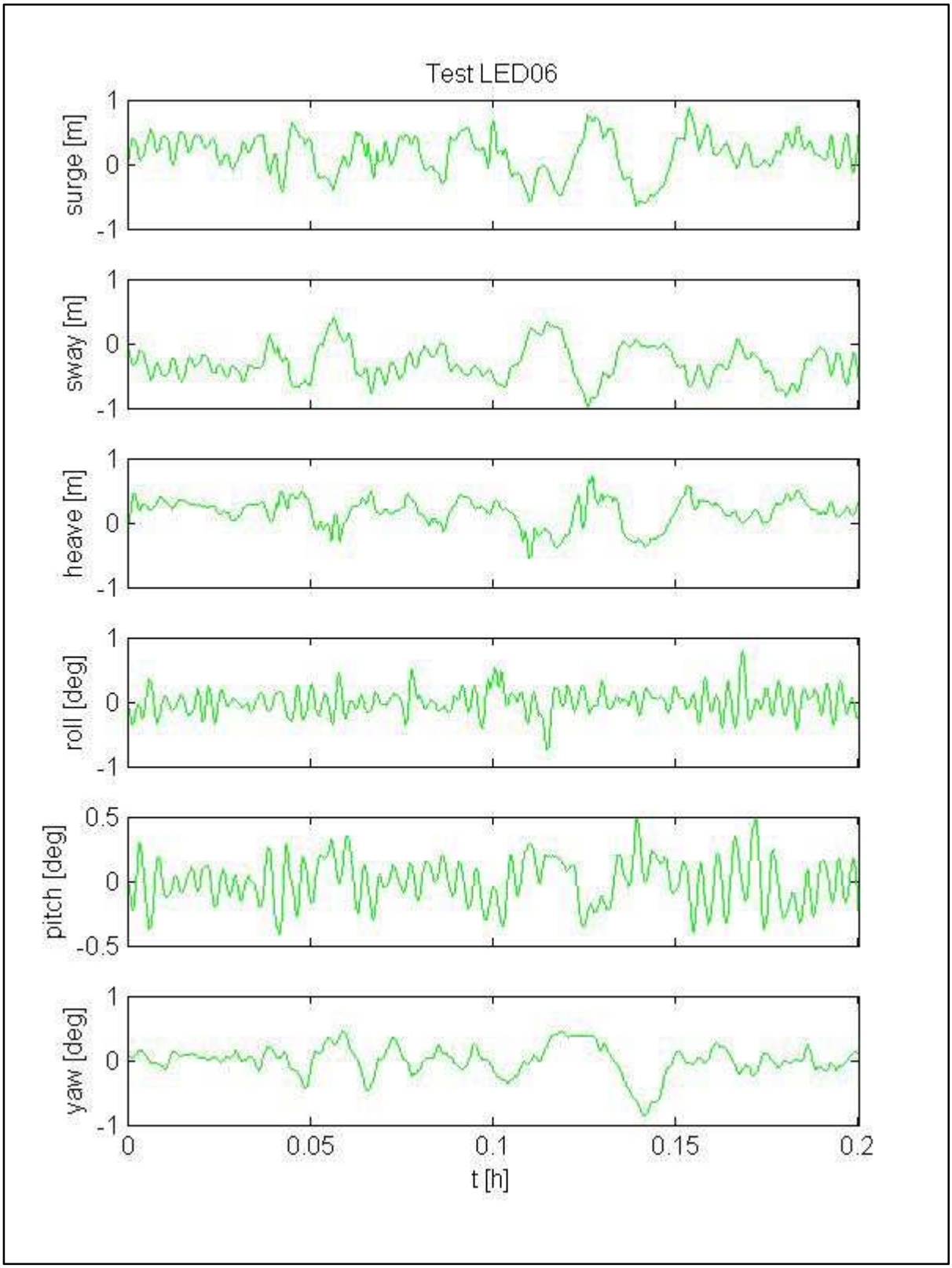

Figure G.6: Motion results for tests LED06

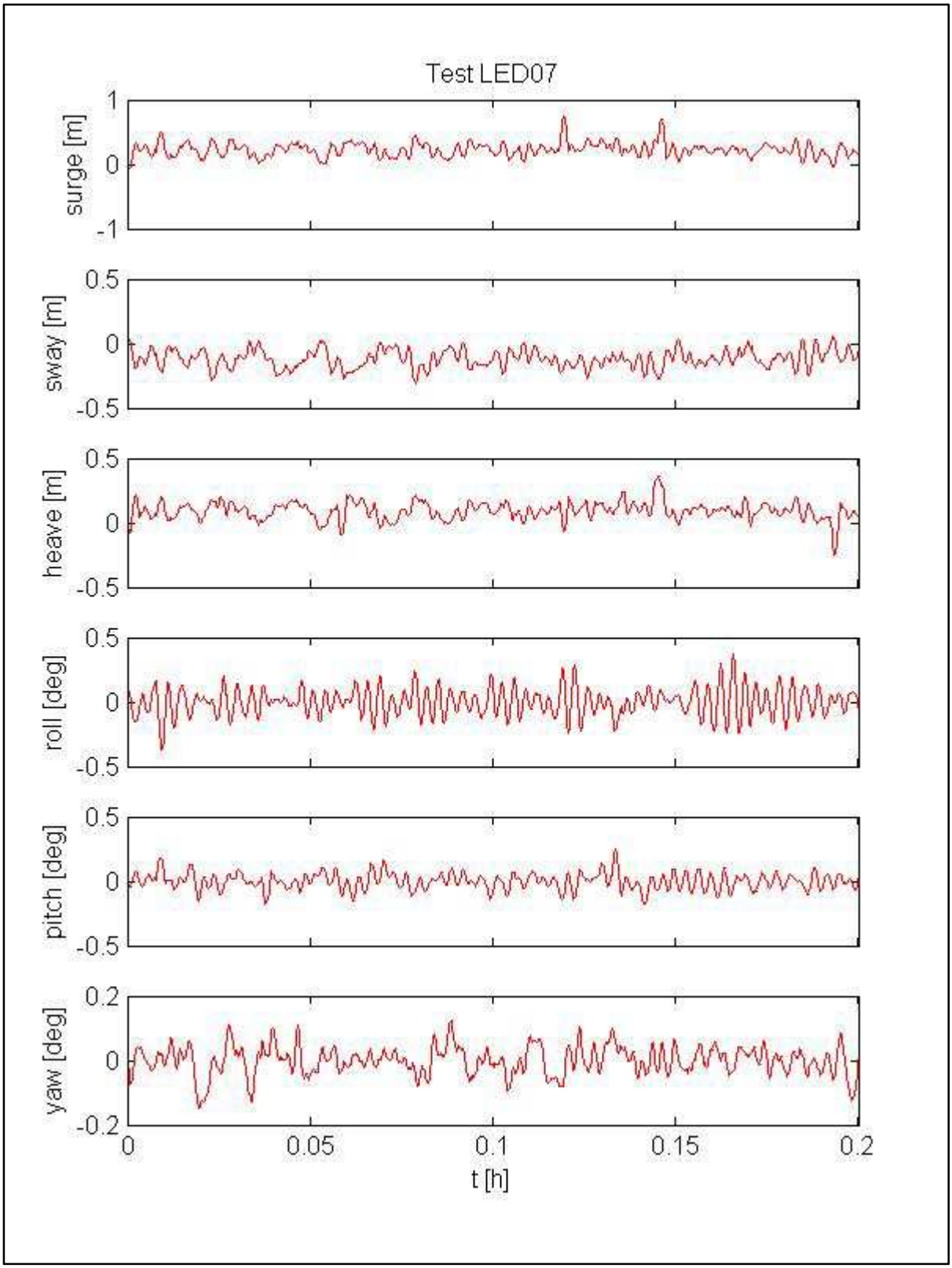

Figure G.7: Motion results for tests LED07

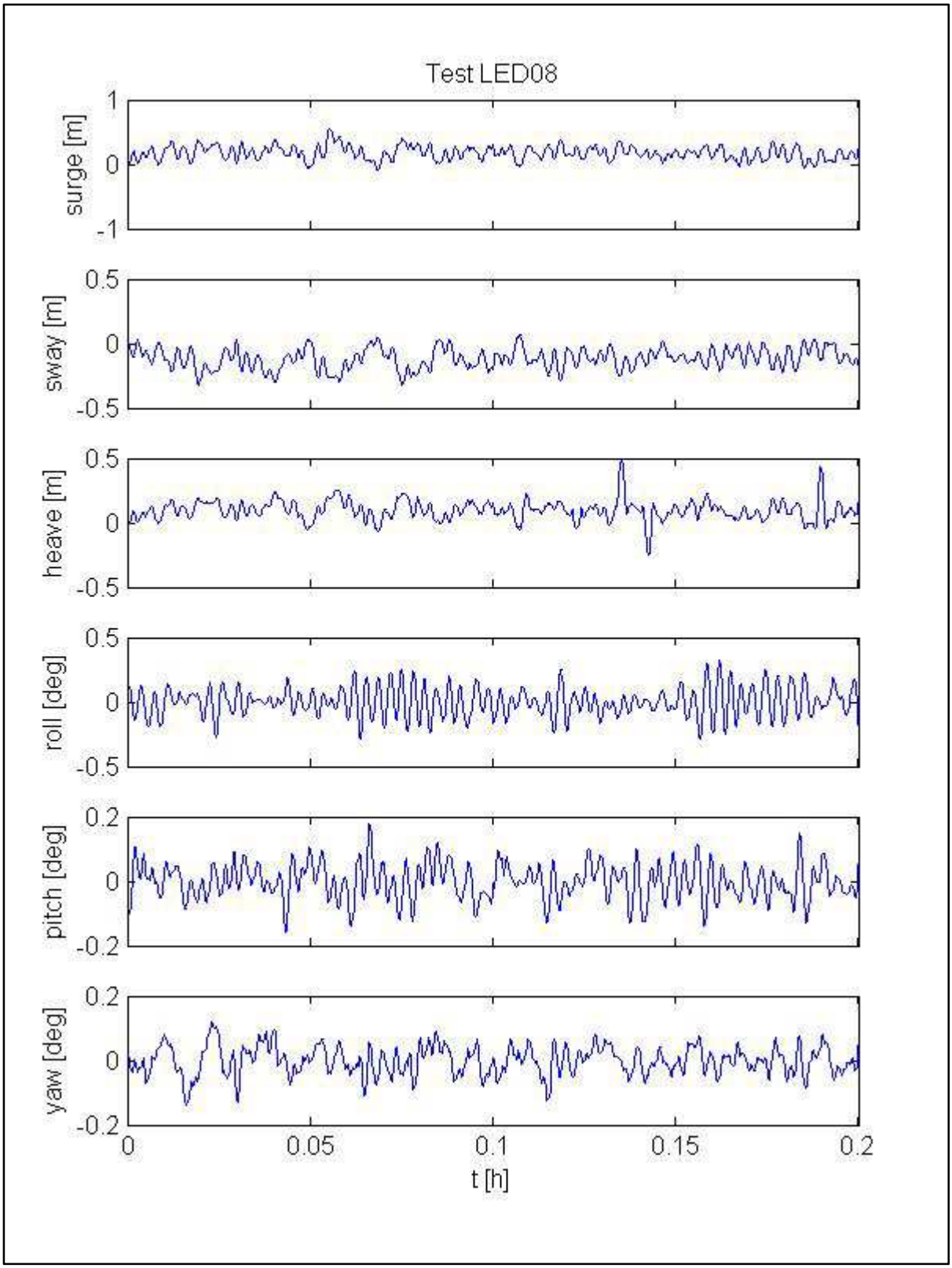

Figure G.8: Motion results for tests LED08

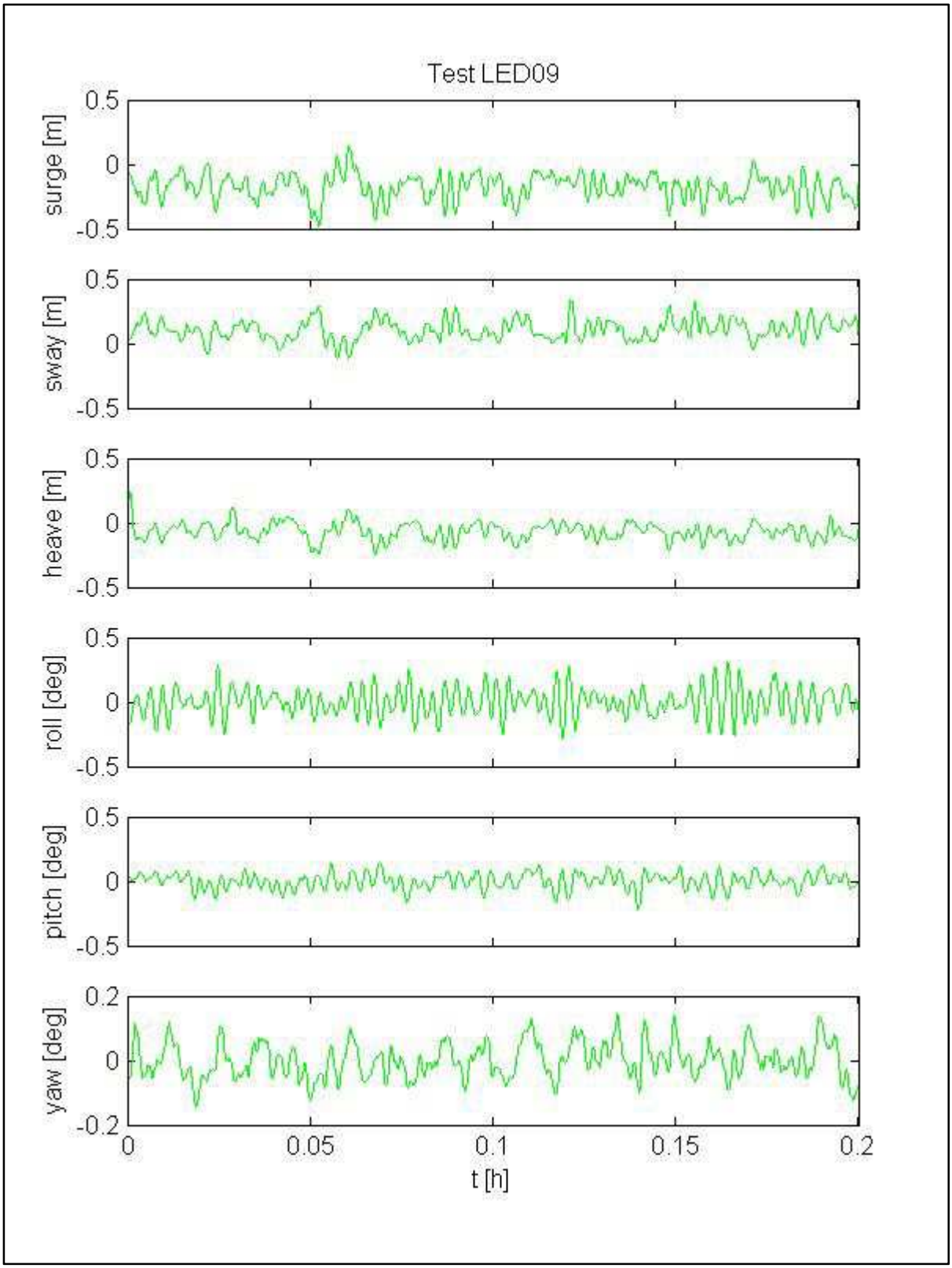

Figure G.9: Motion results for tests LED09

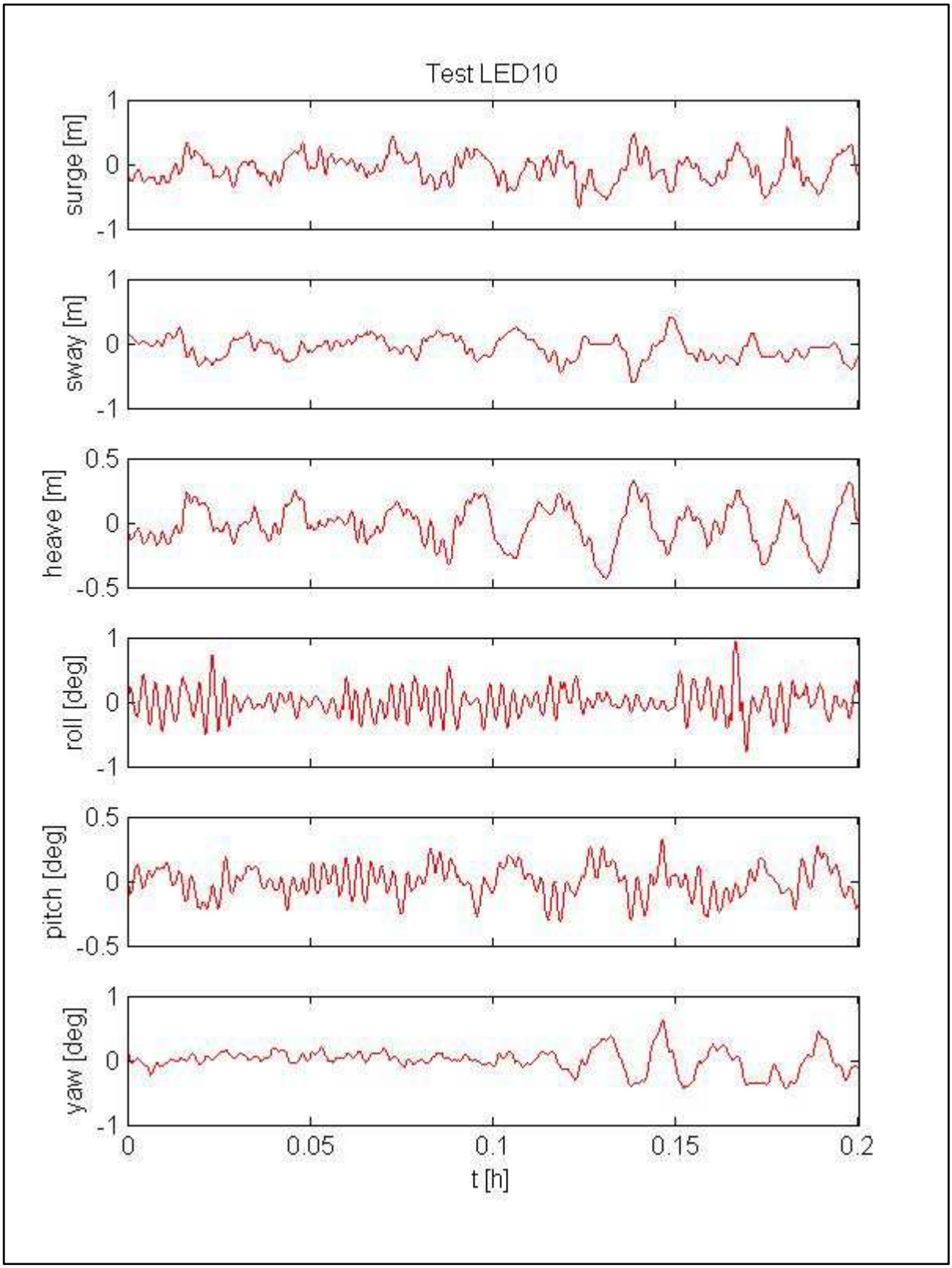

Figure G.10: Motion results for tests LED10

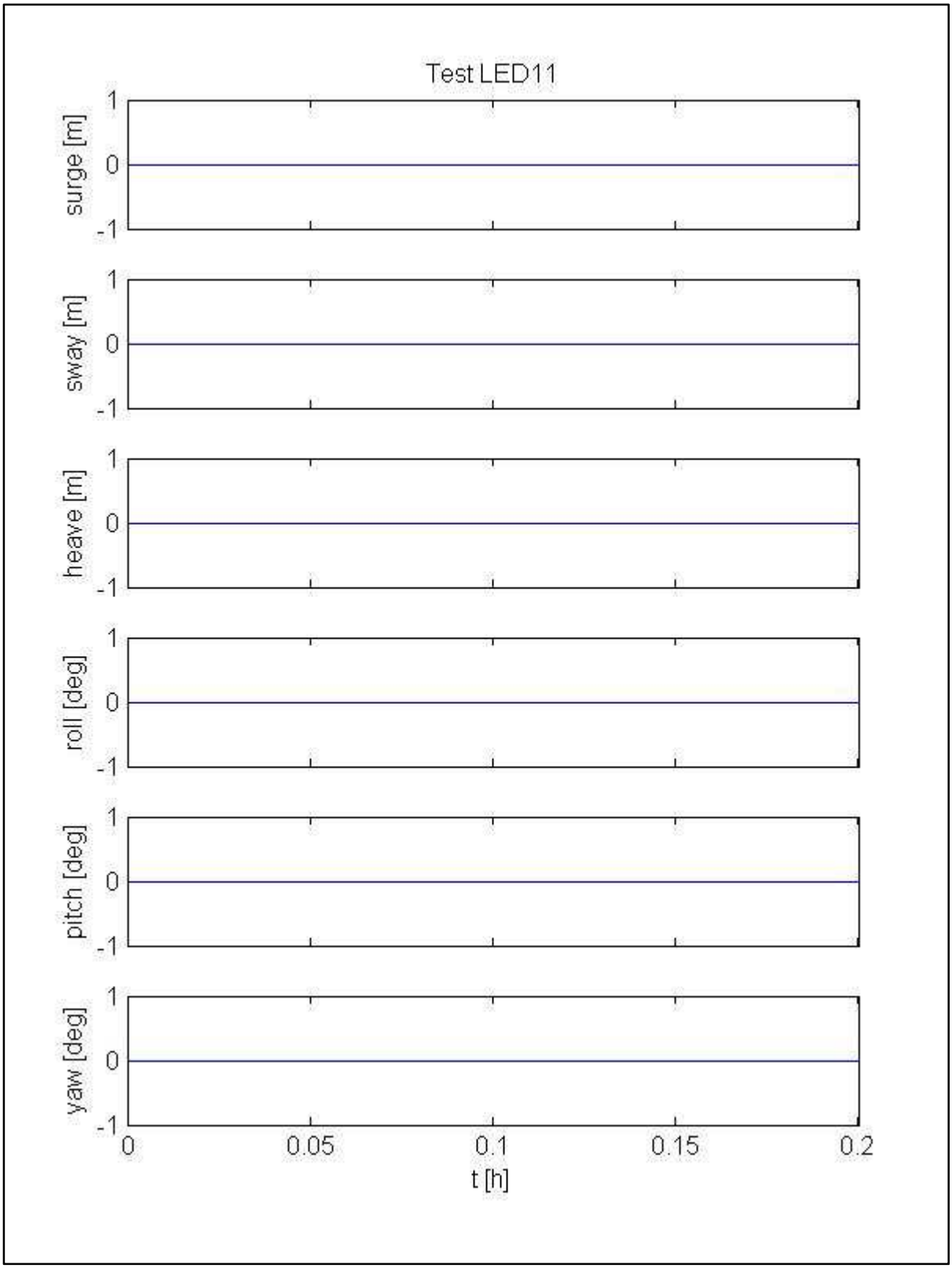

Figure G.11: Motion results for tests LED11

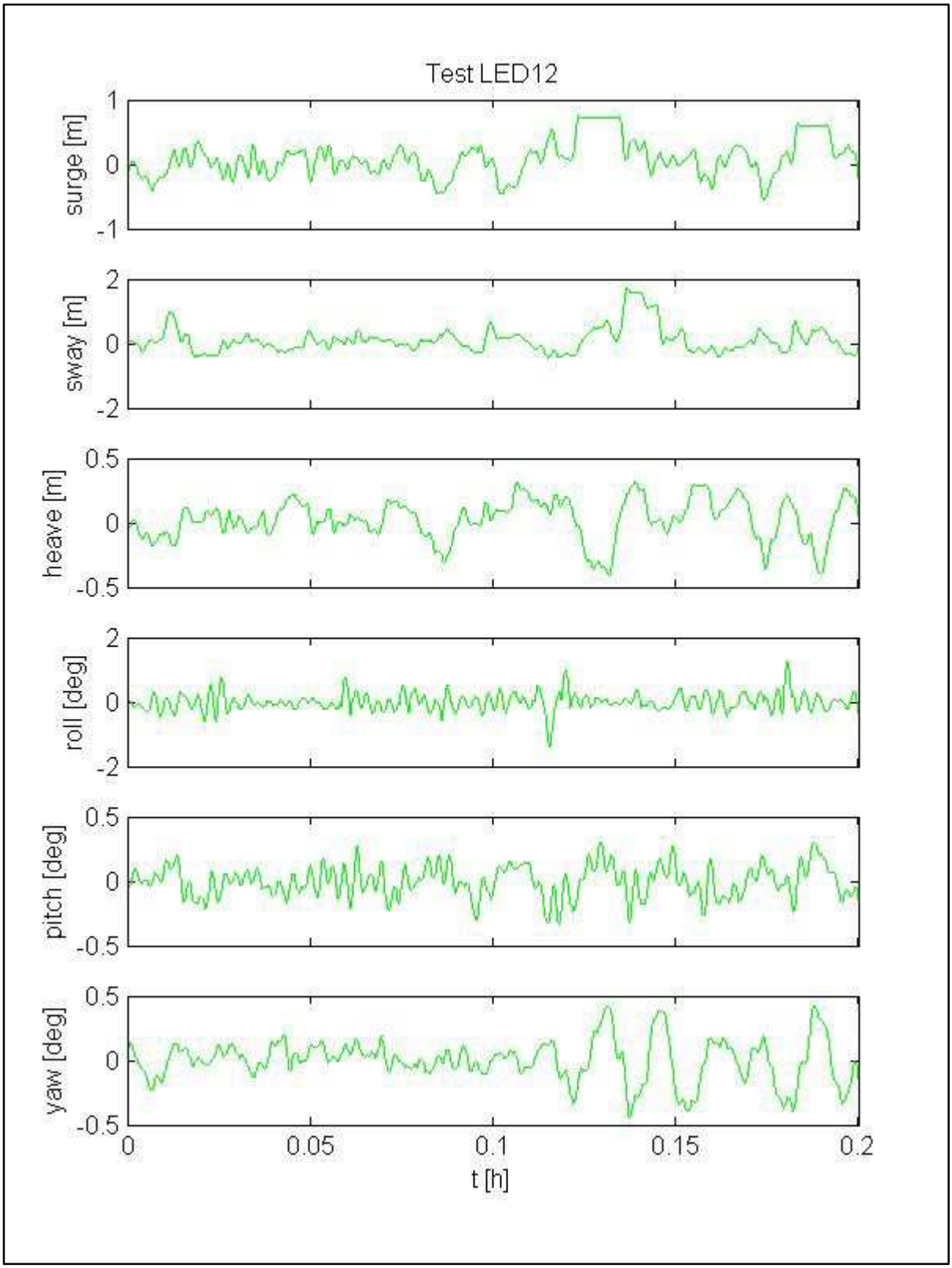

Figure G.12: Motion results for tests LED12

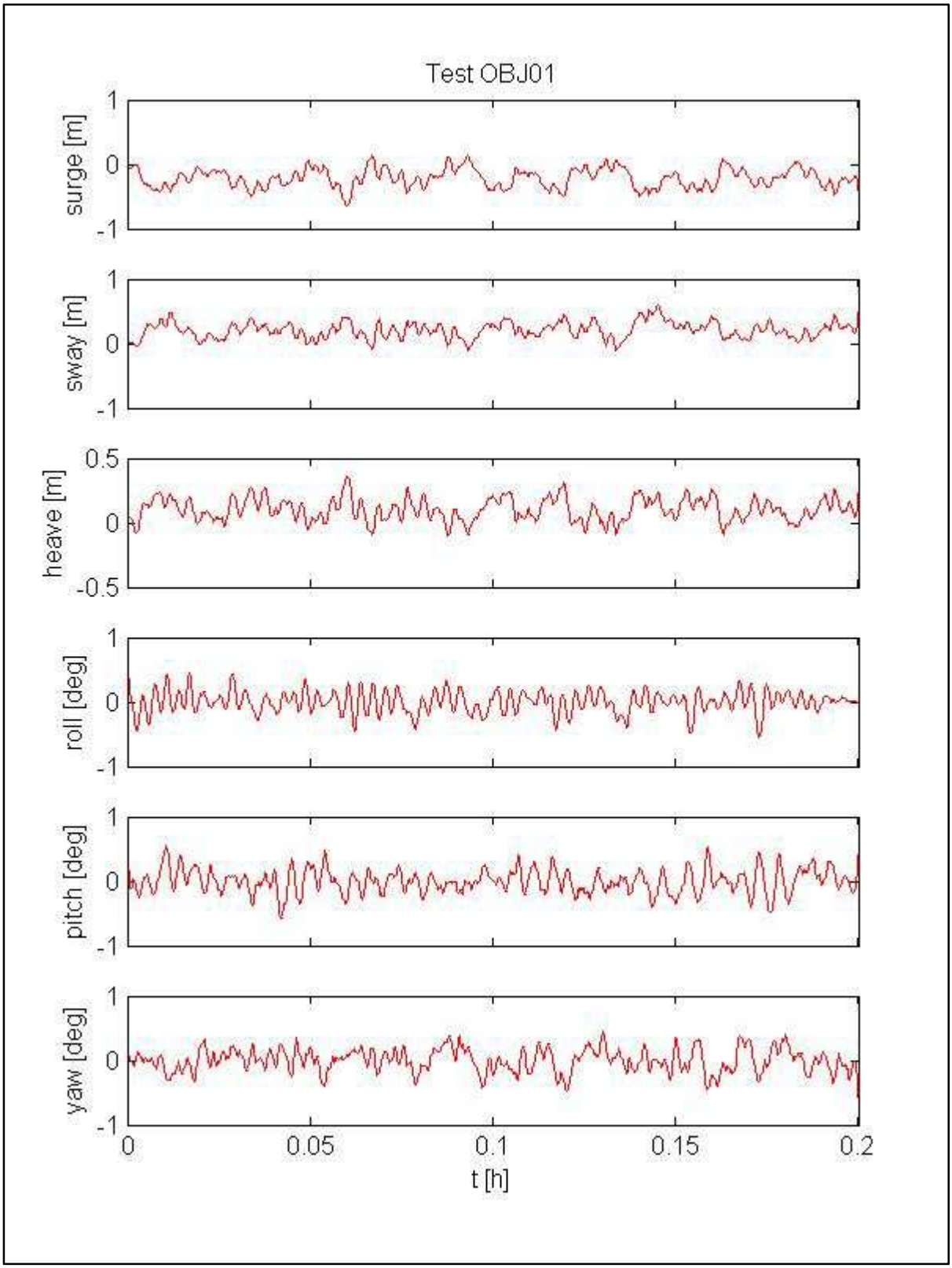

Figure G.13: Motion results for tests OBJ01

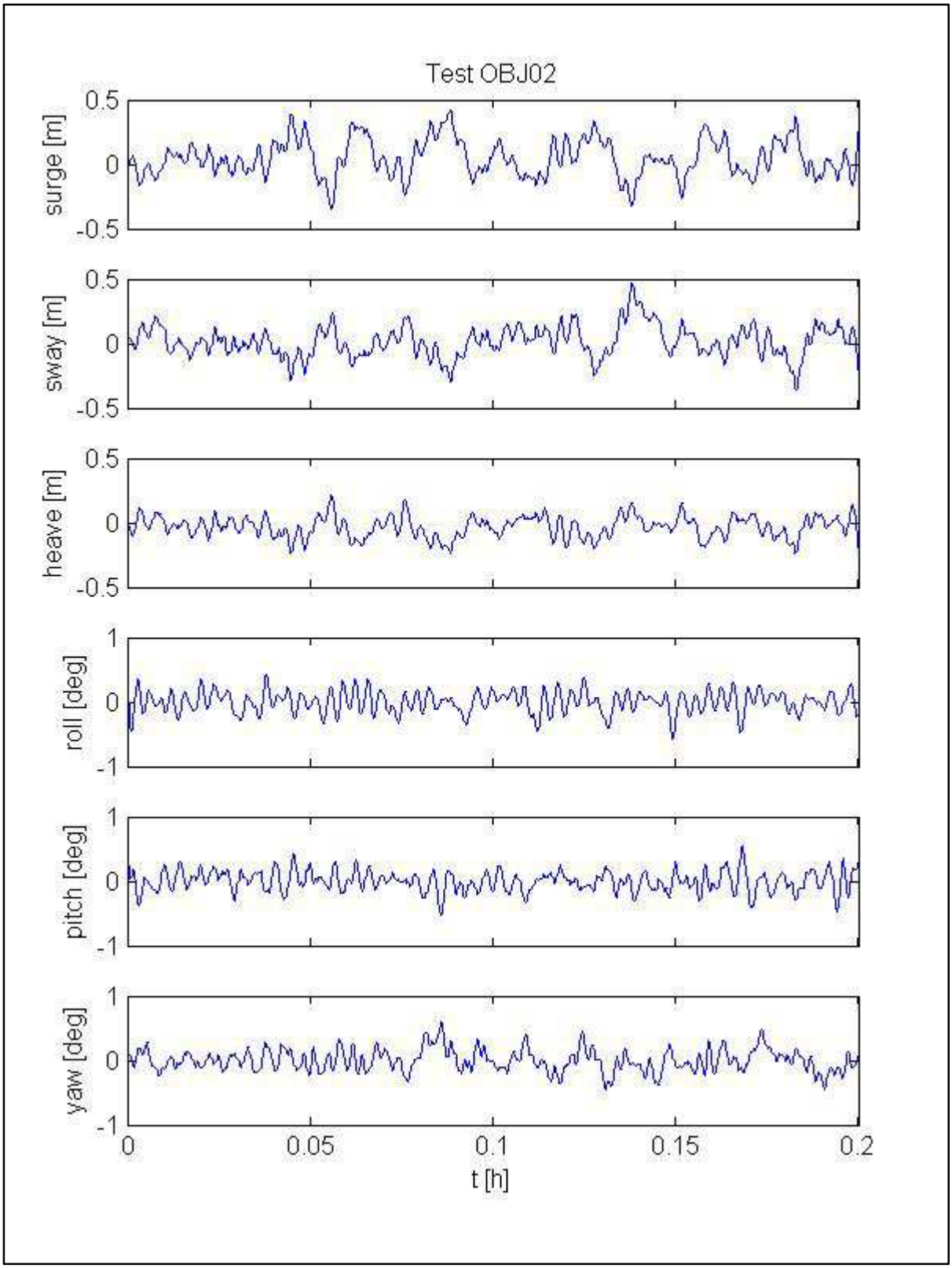

Figure G.14: Motion results for tests OBJ02

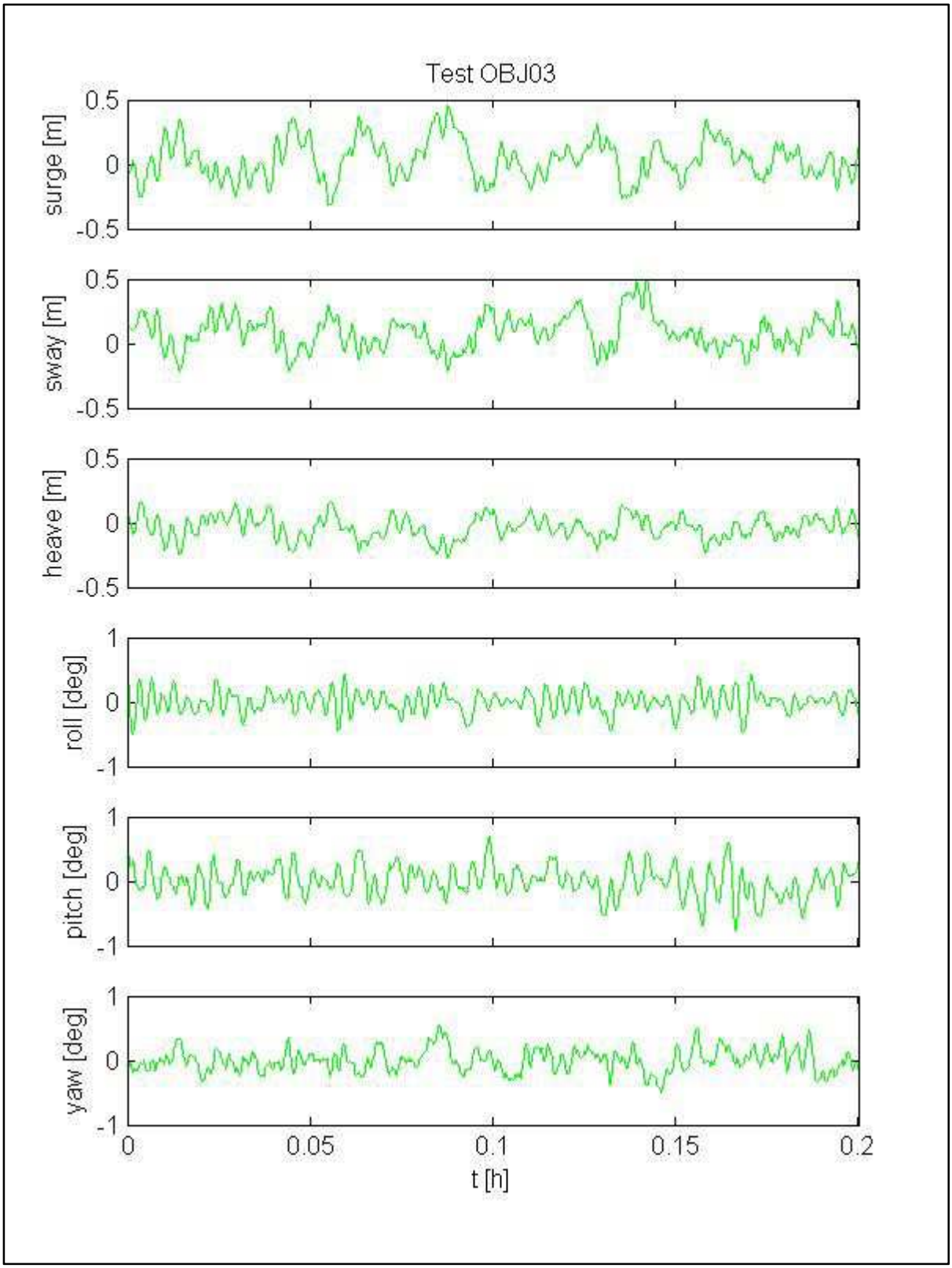

Figure G.15: Motion results for tests OBJ03

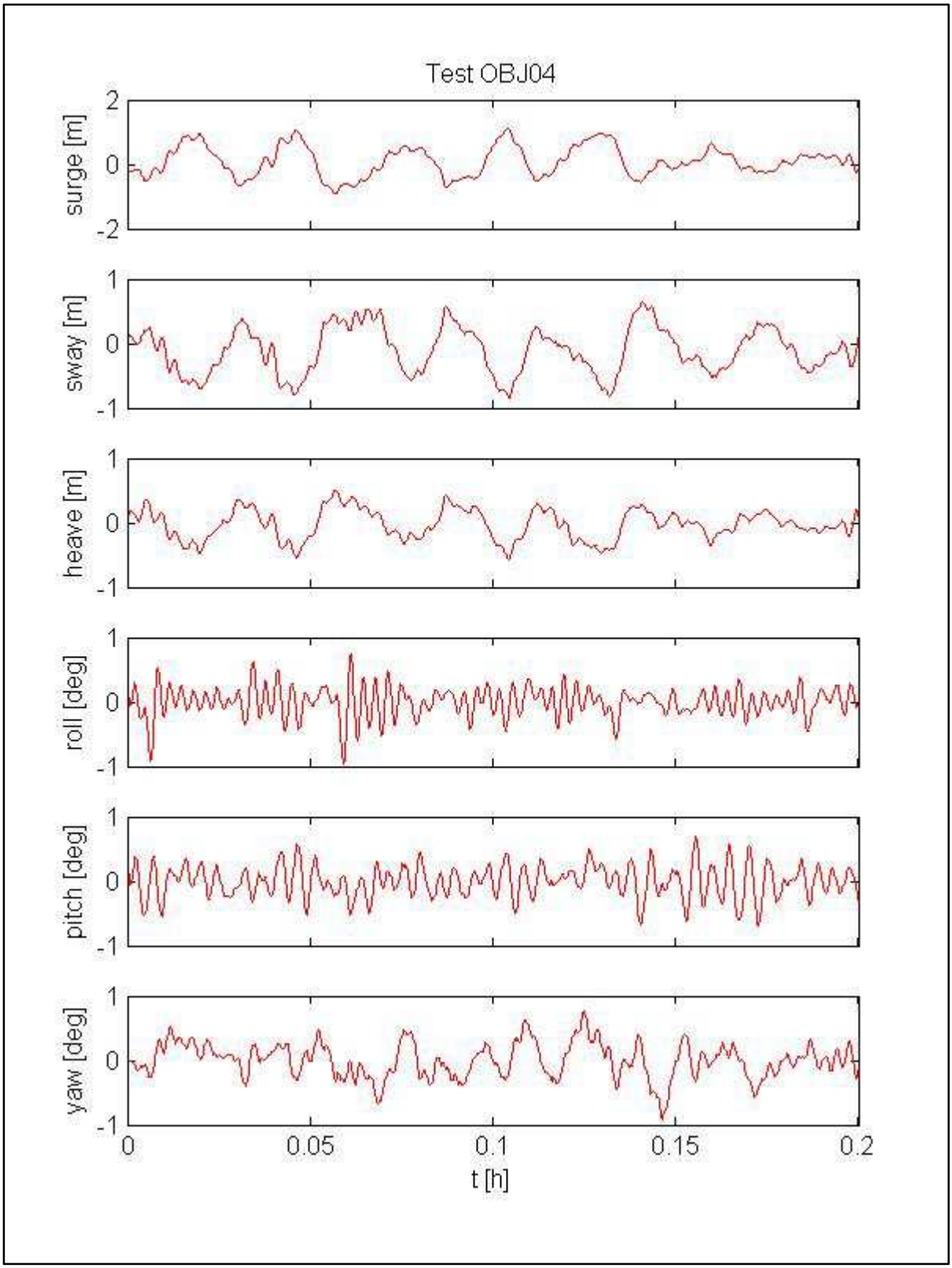

Figure G.16: Motion results for tests OBJ04

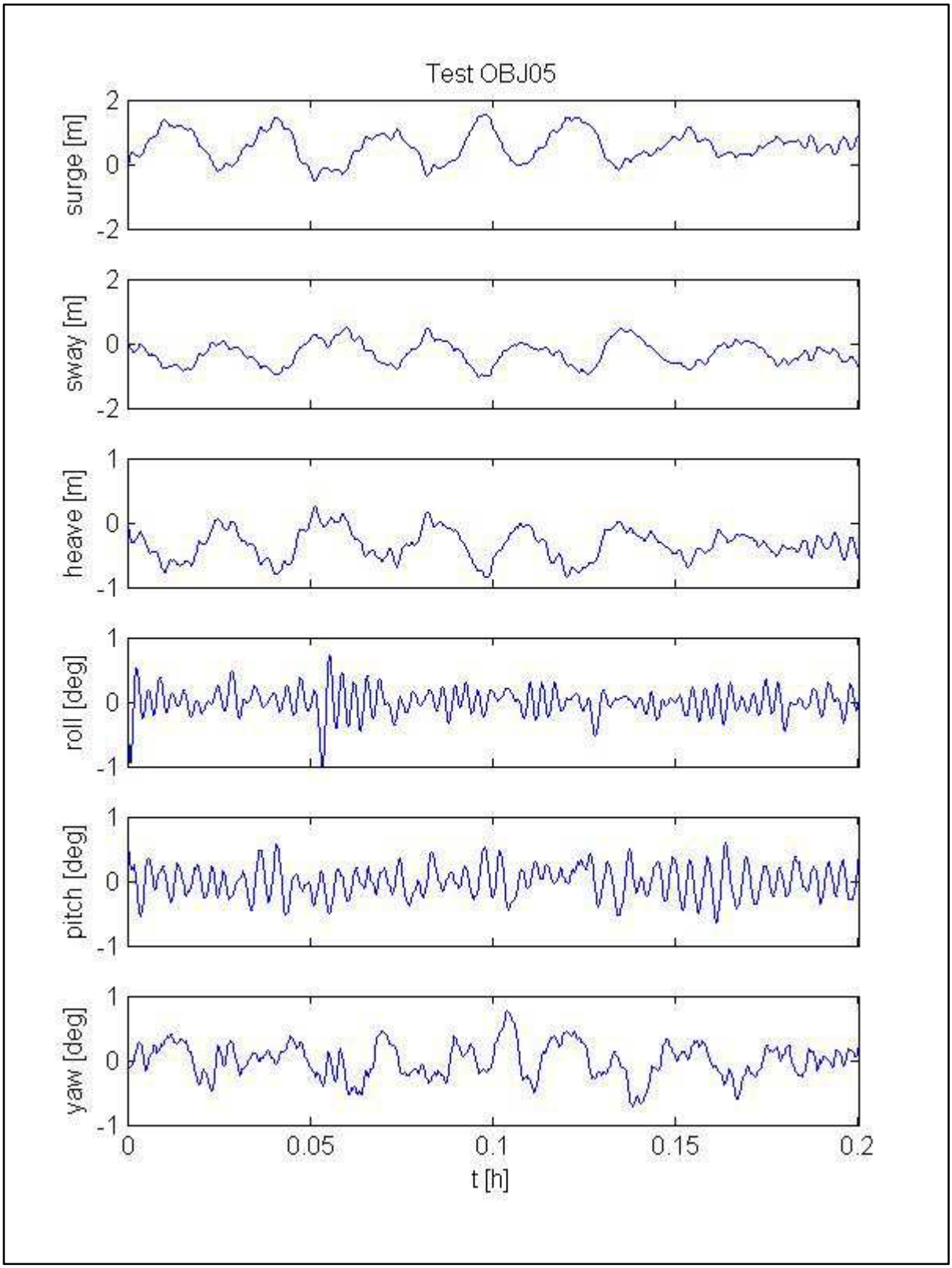

Figure G.17: Motion results for tests OBJ05

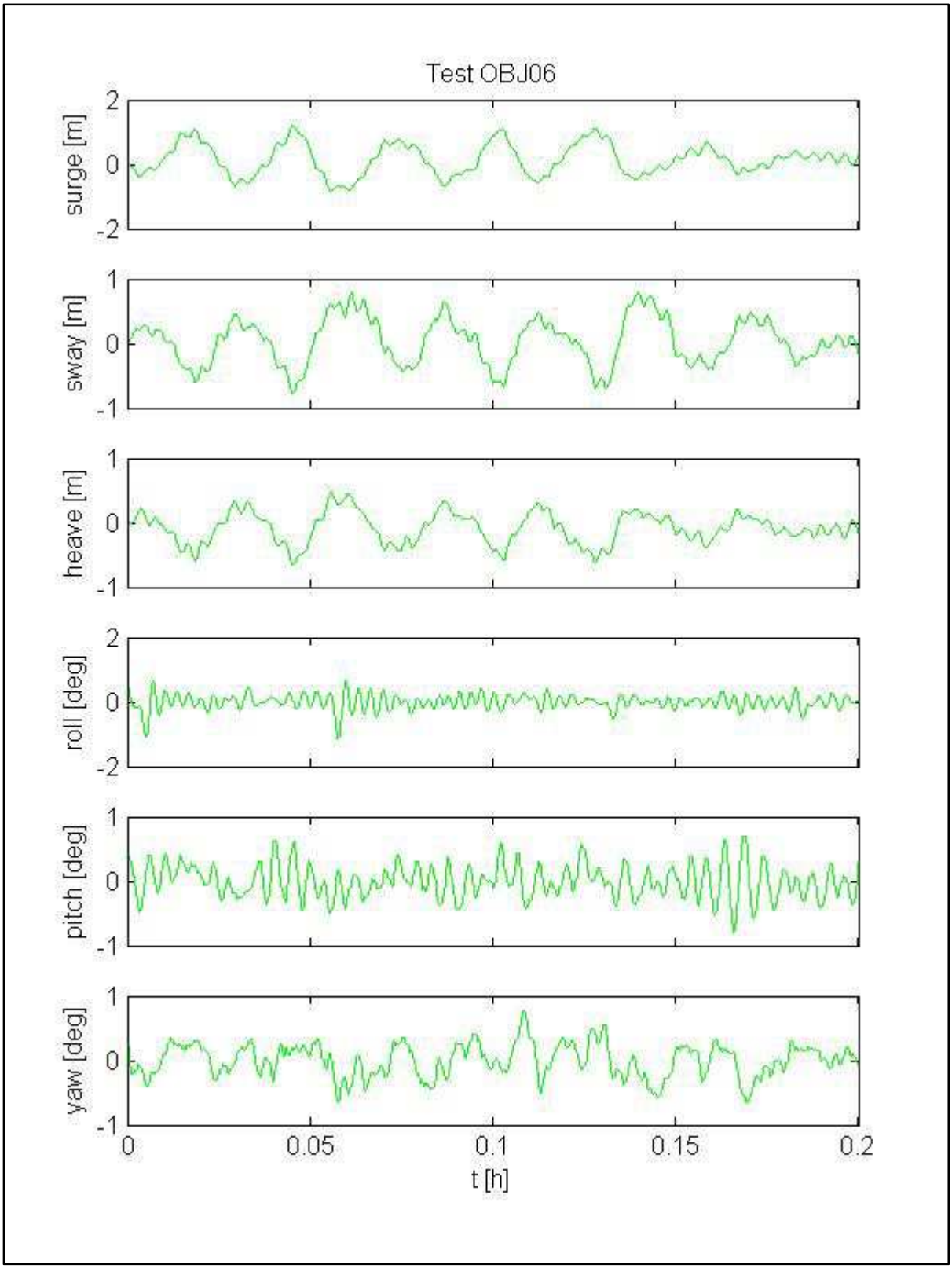

Figure G.18: Motion results for tests OBJ06

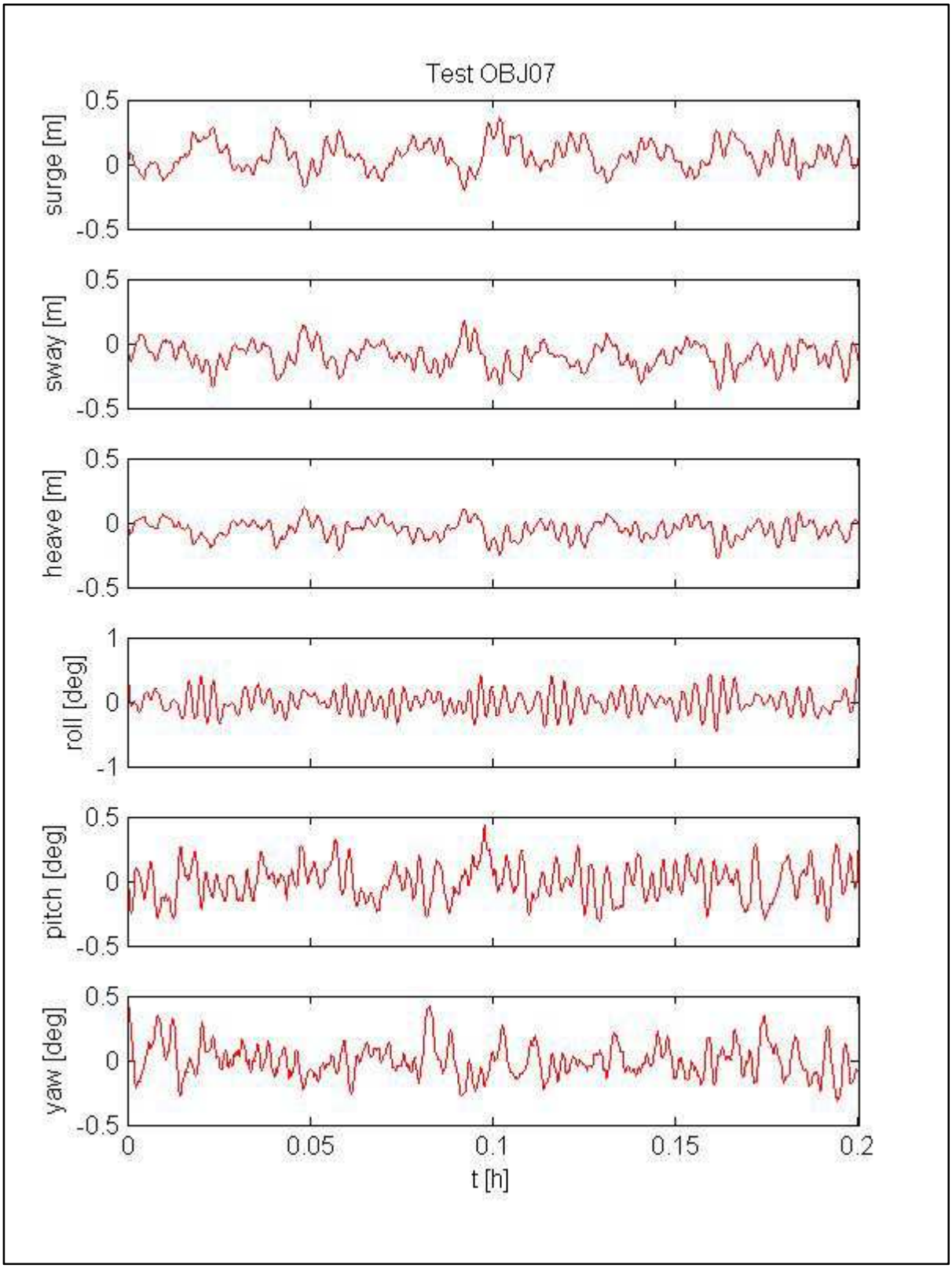

Figure G.19: Motion results for tests OBJ07

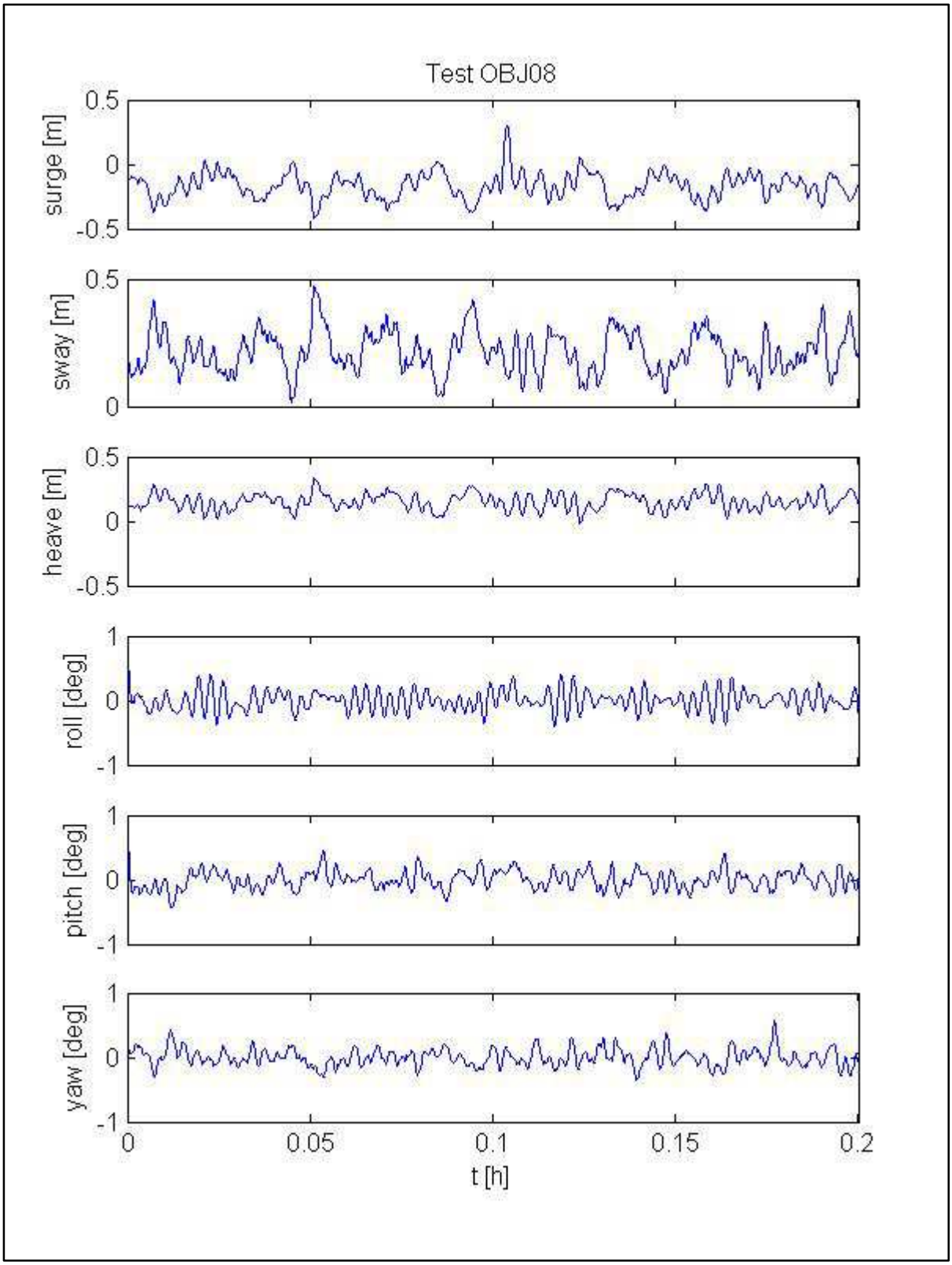

Figure G.20: Motion results for tests OBJ08

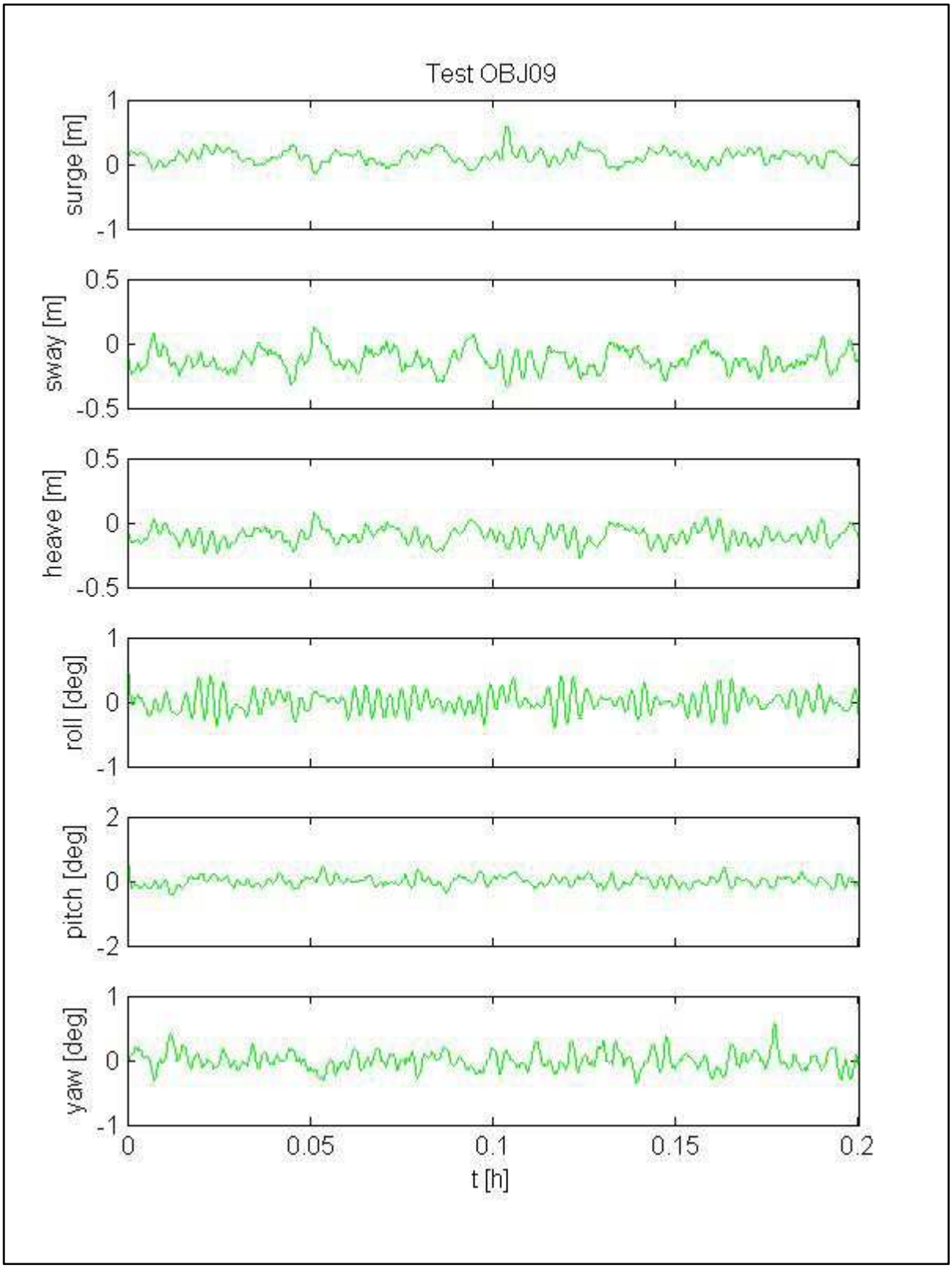

Figure G.21: Motion results for tests OBJ09

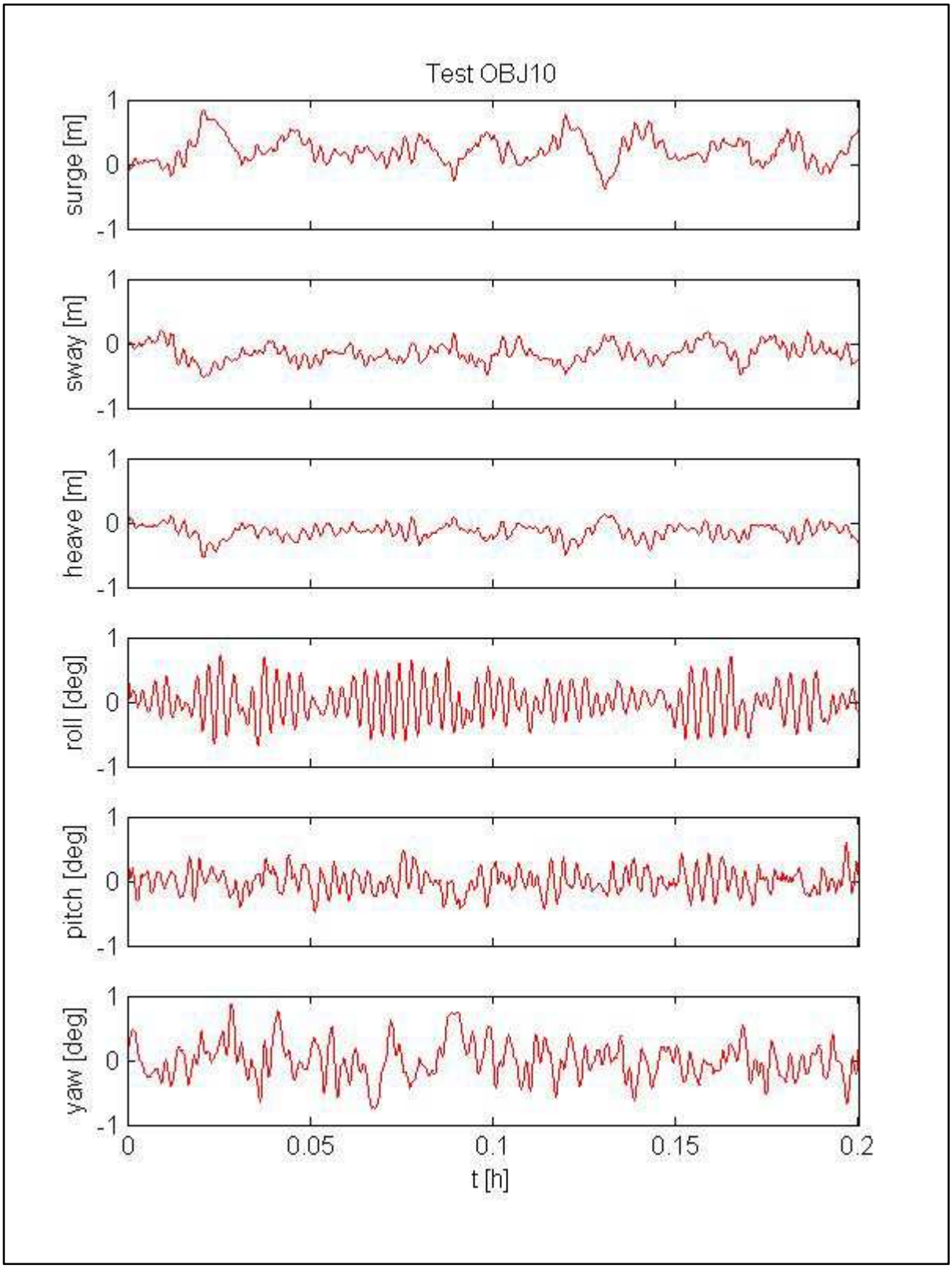

Figure G.22: Motion results for tests OBJ10

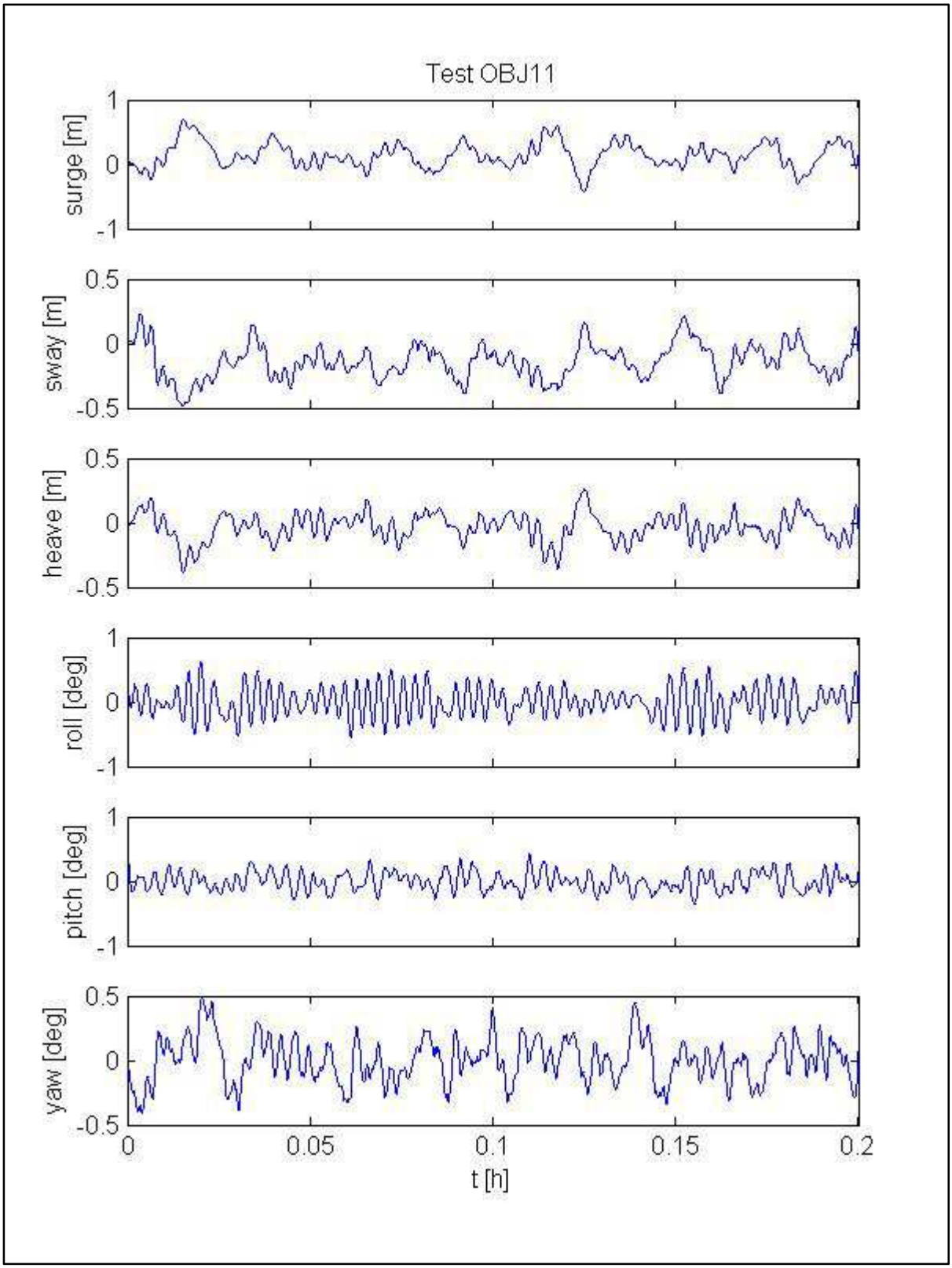

Figure G.23: Motion results for tests OBJ11

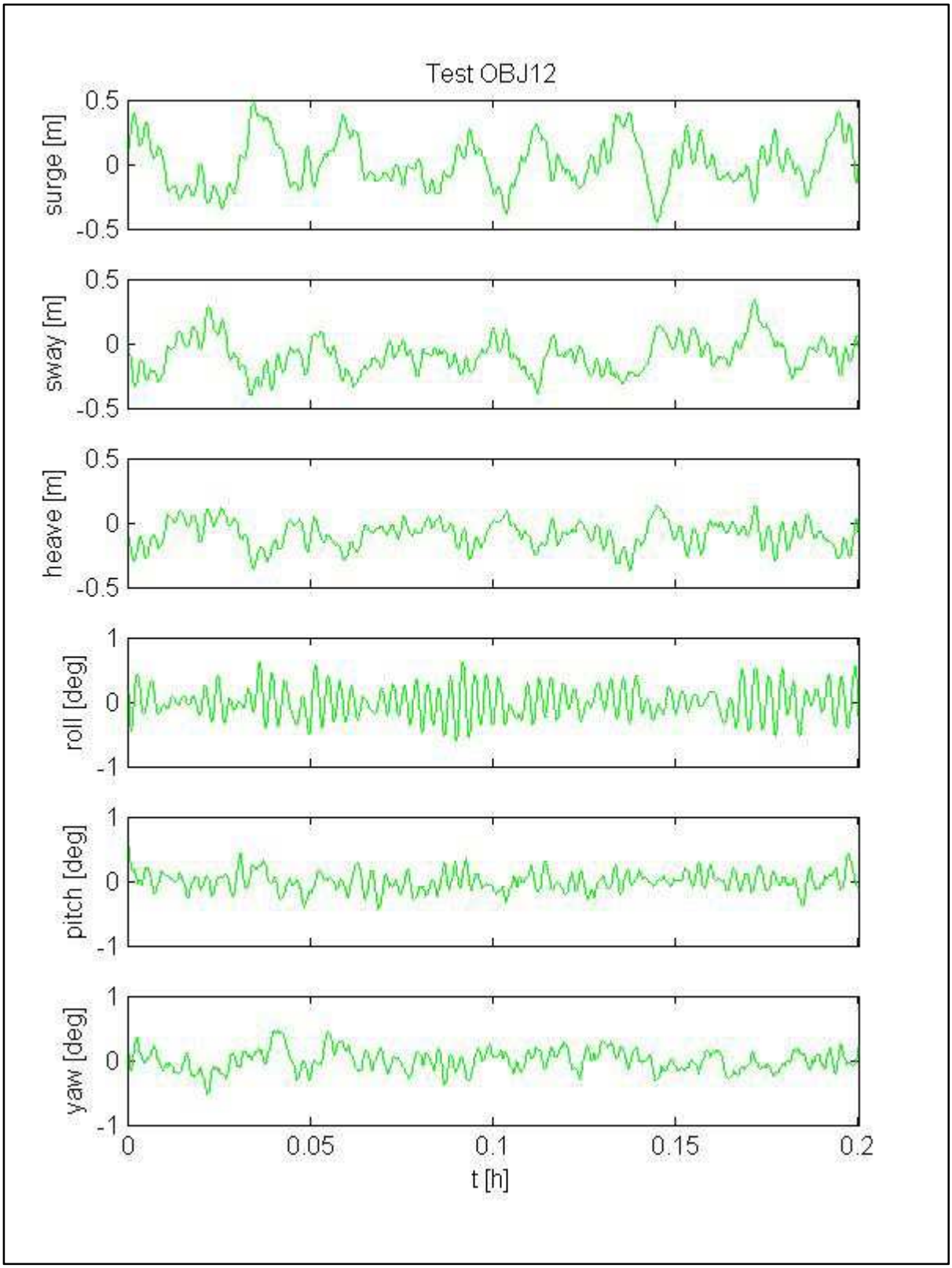

Figure G.24: Motion results for tests OBJ12